Multi-Purpose Data Displays as Wrist Watch Replacement

Peter Hutterer

DIPLOMARBEIT

eingereicht am
Fachhochschul-Diplomstudiengang

Medientechnik und -design

in Hagenberg

im Juli 2004

© Copyright 2004 Peter Hutterer

Alle Rechte vorbehalten

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 15. Juni 2004

Peter Hutterer

Contents

Erklärung iii

Kurzfassung vii

Abstract viii

Acknowledgements ix

1 Introduction and Motivation 1
- 1.1 Description . . . 1
- 1.2 Scenarios . . . 2
- 1.3 Structure of this thesis . . . 2

2 Related Work 3
- 2.1 Commercial Watches . . . 3
  - 2.1.1 Matsucom OnHand PC . . . 3
  - 2.1.2 IBM Linux Wristwatch . . . 4
  - 2.1.3 Fossil Wrist Net . . . 4
  - 2.1.4 Fossil WristPDA . . . 5
  - 2.1.5 Timex USB Datalink . . . 5
  - 2.1.6 Field Technology CxMP Ltd. Smart Watch . . . 5
- 2.2 Problems with previous projects . . . 6
  - 2.2.1 Battery life . . . 6
  - 2.2.2 Processor power . . . 6
  - 2.2.3 User interface . . . 7
  - 2.2.4 Connectivity . . . 8
  - 2.2.5 Extensibility . . . 8
  - 2.2.6 Interaction . . . 8

3 Concepts 9
- 3.1 The Concept of the PersonalServer . . . 9
  - 3.1.1 The PersonalServer Device . . . 10
  - 3.1.2 The Watch Device . . . 11
  - 3.1.3 The PersonalServer Framework . . . 12
- 3.2 The Hardware of the Watch . . . 15
  - 3.2.1 The Display . . . 15
  - 3.2.2 Connectivity . . . 17
  - 3.2.3 User Interaction . . . 19
- 3.3 Switching applications on the Watch . . . 21

3.3.1 Task Manager 21
3.3.2 Interactivity on the Watch 22
3.3.3 Smart Application Switching 22
3.4 Switching PersonalServer 24
3.4.1 Synchronizing Servers 25
3.4.2 Synchronizing Data 26

4 The Implementation 28
4.1 The Java Implementation 28
4.1.1 Why Java? 28
4.1.2 The Plugin Framework in Java 29
4.1.3 The Application Template 30
4.1.4 The Java PersonalServer 31
4.1.5 Problems with Java 32
4.2 Linux on the iPAQ 33
4.2.1 Reasons for Linux 33
4.2.2 Running Linux on the iPAQ 34
4.3 The C Implementation 35
4.3.1 Why C? 35
4.3.2 The Plugin Framework in C 35
4.3.3 The Application Template 36
4.3.4 The C PersonalServer 37
4.4 The Image Transfer Protocol 40
4.5 Example Applications 41
4.5.1 Direct Hardware Communication - The Timer 42
4.5.2 Communicating to other Applications - MP3 Title Display 43
4.5.3 Live Data Streams - RSS Feeds 44
4.5.4 Personal Information Management - Alarm Clock 46
4.5.5 Video Streaming - Baby Monitor 47

5 The Prototype 48
5.1 The Hardware 48
5.1.1 The Display 48
5.1.2 The Bluetooth Chip 49
5.1.3 The Microcontroller 49
5.2 Code on the Watch 50
5.2.1 Setting up the Watch 50
5.2.2 Displaying Data 51
5.3 Issues with the Watch 52
5.3.1 Processor speed 52
5.3.2 Processor memory 52

6 Review and Future Works 54
6.1 Critical review 54
6.2 Future Works 55

A CD-ROM Content **56**
A.1 Thesis 56
A.2 Online resources 56
A.3 Implementation 56
A.4 Additional software 57

Bibliography **59**

Kurzfassung

Eine Vielzahl an Projekten beschäftigte sich die letzten Jahre intensiv damit, Armbanduhren um komplexere Funktionen zu erweitern. Das Spektrum der Entwicklungen reicht von speziellen Uhren mit Radioempfang bis hin zum Mini-PDA am Handgelenk.

Die innovative Neuerung, die im Rahmen dieses Projektes ausgearbeitet wurde, ersetzt eine herkömmliche Armbanduhr durch ein universell anwendbares Datendisplay. Die Uhr ist somit nicht mehr auf proprietäre Funktionalitäten beschränkt. In Verbindung mit einem Handheld Computer, welcher als Server fungiert, ergeben sich nahezu unbeschränkte Möglichkeiten der Applikationsnutzung.

Die vorliegende Arbeit beschreibt detailliert Anforderungen bezüglich Hard- und Software sowie die Konzepte für ein solches Projekt. Überdies wird der Prototyp vorgestellt, der eigens für diese Zwecke entwickelt wurde. Als Handheld wurde ein HP iPAQ gewählt, auf welchem die Serverlogik implementiert wurde. Dieser sogenannte PersonalServer bedient sich der Bluetooth Technologie um sich kabellos zu der Uhr zu verbinden. Durch das Auslagern der performance-intensiven Berechnungen auf den PersonalServer wird die Uhr selbst entlastet und der Aufgabenbereich auf die reine Datenvisualisierung beschränkt. Dies erlaubt es die Hardware des der Uhr einfach zu halten aber dennoch komplexe Applikationen auszufhren. Zudem wird eine Integration der Uhr als zusätzliches Display für bestehende Anwendungen stark vereinfacht.

Abstract

In recent years more and more projects have tried to bring complex functionality to the watch, which up until now has mostly been limited to displaying the time. The spectrum now ranges from miniature PDAs on the wrist to watches able to receive FM radio waves.

This project is an attempt to replace the wrist watch with an all-purpose data display. By using such a data display the watch is not limited to a certain set of applications as it is the case in most other projects. Paired with a handheld computer acting as server for the watch nearly any application can be incorporated into the display.

This thesis describes the concepts and the hard- and software requirements for such an approach. Furthermore, a prototype is introduced which was developed by following those concepts. An HP iPAQ is used as handheld computer in the implementation, the watch itself was purpose-built for this project. The iPAQ runs the "PersonalServer" which connects over Bluetooth to the watch. By using this PersonalServer the watch can offload tasks and only needs to display incoming data. This allows it to keep the hardware on the watch simple. As most of the software runs on the iPAQ, complex applications can be executed and an integration of the watch as additional display into existing applications is eased.

Acknowledgements

Several people made this project and the thesis possible and I want to thank all of them. First, Michael Haller, my supervisor in Austria, for his support while writing this thesis. Second, Bruce Thomas for giving me the chance to do my internship at the University of Australia and for helping me with this thesis. The time in Australia was the greatest I have had in my life so far.

I want to thank Wayne Piekarski, Benjamin Close, Aaron Toney and the others from the Wearable Computers Lab for their help and patience when I was implementing the prototype. Thanks to Mark Smith and John Ankcorn from the HP Labs in California for developing the watch prototype and providing code and documentation for me.

Finally, I want to thank my parents who made it possible for me to study and of course my friends for pulling me away from the computer from time to time.

Chapter 1

Introduction and Motivation

Watches have been around us for several centuries. As the amount of personal time gets less every year it is more and more important to have a watch around at all times. In the 19th century the watch moved from the pocket to the wrist, thus being visible at all times [43]. Since World War One this move is sealed and nowadays hardly any other locations are used for carrying a watch. Digital watches came up with additional functions such as stopping the time and setting alarms. Ignoring those few exceptions which actually provide compasses, phone books or other additional functionality the purpose of the watch is still mainly (and often solely) to show the time.

Several years ago Personal Digital Assistants (PDA) came up. They seem to get more important every year and are widely used in the business user domain. Those little computers already have more performance than a desktop PC had just a few years ago and they are not limited to Personal Information Management (PIM) anymore. Video and office applications as well as games are standard and even telephony and Augmented Reality [36] is possible with the newest models. Up to now, PDAs are carried around in pockets, just as it was with the watch before it moved on to the wrist. Therefore, it might happen that PDAs in the future will be carried on the wrist and replace the watch. This thesis describes an approach to achieve this goal.

1.1 Description

Several projects exist focusing on the "computer on the wrist" idea, all trying to create a small, fully functional computer and putting it on the wrist. However, this causes several problems as will be explained in Section 2.2. The approach used in this thesis is to use a brainless data display paired with a PDA. By using the PDA as main device it should be possible to use the watch as display for complex applications. As a part of this thesis, a prototype was developed to proof this concept.

Both devices, watch and PDA, have their pros and cons. By not turning down the PDA but instead using it as main device the advantages of both devices sum up. The watch itself does not run complex code and works as an

external data display for the PDA. With this close relationship, the manufacturing costs for the watch are lower and development time for applications is shorter. Moreover, instead of being a independent device the watch can be seen as an additional gadget for the PDA.

The high computing power and the operating systems of today's PDAs allow it to bring complex applications to the watch which would not be possible if the watch was driven by a small microcontroller. Additionally, existing applications already installed on the PDA can be extended to bring data to the display.

The whole project consists of two parts: One part is the watch itself, the hardware and software, used on this wrist device. The other part is the software necessary on the PDA, which allows it to use the watch as data display.

This thesis will discuss the benefits and drawbacks of the close relationship between watch and PDA. Furthermore, it will show that the approach to use a brainless display has usability drawbacks which can only be avoided by using more autonomous devices.

1.2 Scenarios

Possible scenarios for a multi-purpose watch can be found everywhere. SMS or video telephony could be displayed on the watch, as well as text messages from instant messaging systems. Moreover, for business users several applications are interesting, such as live stock prices, meeting schedules or to-do lists.

By allowing different devices to access the watch it can be used in a home environment: Kitchen devices could send information about ongoing tasks, the TV could display the current channel's program or the radio the current playlist.

Since the PersonalServer runs on an external device, it is possible to switch servers on the fly. Different sets of applications can be used on the same watch by changing the server, even several watches could be serviced by just one server. This might be interesting for a company which wants to equip its employees with the watch.

1.3 Structure of this thesis

This thesis gives an overview about a project using a brainless data display instead of a watch on the wrist. First, Chapter 2 analyzes existing projects using a computer as a wrist device. Chapter 3 explains the concepts behind the project and points out what is different to existing approaches. It describes the software requirements and which hardware is needed. The actual implementations are shown in Chapter 4 where some example applications are described too. Additionally, the prototype of the watch, developed at the Hewlett Packard Labs in California, US, is described in Chapter 5. Finally, Chapter 6 shows a view about the possible future of intelligent wrist watches. Appendix A lists the contents of the CD provided with this thesis.

Chapter 2

Related Work

Several projects have targeted to replace the watch with a multi-functional device. Some research work has been done about the social weight of a watch device in [61]. Additionally, the authors of [47] ask the question what applications an all-purpose watch could be used for.

This chapter will give a short overview about some of the watches on the market right now and point out their features. However, this is a very small selection of complex watches since listing every watch is beyond the scope of this thesis. Problems occurring with those projects will be discussed in Section 2.2.

2.1 Commercial Watches

An overview over six different devices is given now. Five are watches available in online stores, the IBM Linux Wristwatch is just a research project. All six have slight differences in functionality, the Matsucom OnHand PC and the IBM Linux Wristwatch try to create a miniature computer on the wrist. Fossil's Wrist PDA is an approach to create a miniature PDA device whereas the Wrist Net uses a very selective channel based data set. The Timex USB Datalink tries to fill the gap between sports watches and PDAs, the main focus is still on the watch functions. The Field Technology CxMP Smart Watch is the only commercially sold watch in this list with a color display and supports image viewing and even sound output.

2.1.1 Matsucom OnHand PC

The Matsucom OnHand PC as shown in Figure 2.1 was introduced in 1999 [60] and has a monochromatic display of 102x64 pixel. A set of over 30 programs is preinstalled, containing different clock faces, a world clock, PIM features and even games. The OnHand PC is running W-PC-DOS as operating system and provides a rich and well documented API for programmers.

Synchronization is done via a serial cradle and a special software. Wireless connections are possible via an infrared interface.

The OnHand PC is also manufactured by Seiko under the name "Ruputer". However, it never got more than the status of a toy.

Figure 2.1: The Matsucom OnHand PC. Image taken from http://www.matsucomusa.com/Download/Media/onHand.jpg. Copyright by Matsucom, Inc.

2.1.2 IBM Linux Wristwatch

IBM tried a different approach with the Linux Wristwatch. The developer team wanted to build a wrist computer with a standard operating system instead of one optimized for embedded devices. Finally, they ended up using Linux with slight software adaptations for the processor. The graphical interface uses the X standard just as a desktop linux system uses it. Some modifications had to be made though to fit the X server onto the watch.

The watch consists in two version (see Figure 2.2), the first, earlier version, has a 96x120 LCD in portrait format, the second version a VGA OLED in landscape format. Both screens are touchscreens, divided into four parts which act as buttons. Additionally, both watches have a jog-wheel.

The watch provides a Bluetooth and infrared interface for connectivity and a microphone and a small speaker for audio input and output.

2.1.3 Fossil Wrist Net

Fossil and Microsoft are trying quite a different approach with the MSN Wrist Net. The goal is not to have a miniature computer on the wrist but instead more the digital equivalent of a radio. This watch can fetch information via FM radio waves from preconfigured channels. No other connectivity is built in than those radio waves.

This is naturally not as personalized as on the other devices and no other programs are installable either. The only way to get information is to use one of the channels. At the time of writing this thesis, subscribing to a channel costs USD 9.99 per month or USD 59 per year [28]. However, there is personalized information available too: receiving MSN messenger messages and receiving schedules over a plugin for Microsoft Outlook are supported. Due to the radio transmission the delivery of the messages cannot be guaranteed in real-time [57].

Fossil has three different models of this watch, other companies such as Suunto are manufacturing their own watches based on the same technology. The watches are a part of Microsoft's Smart Personal Object Technology (SPOT) initiative [14].

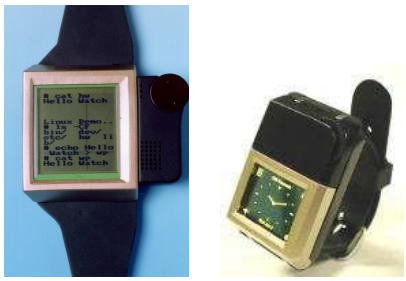


Figure 2.2: The IBM LCD Linux Wristwatch (left) and the OLED version (right). Images taken from http://www.research.ibm.com/WearableComputing/factsheet.html. Copyright by International Business Machines Corporation, Inc.

2.1.4 Fossil WristPDA

Quite an old model is Fossil's Wrist PDA. This small computer is using PalmOS 4.1 and therefore the PIM applications known from Palm PDAs. The interaction with the device is done over a touchscreen with a stylus, just as it is common for normal-sized PDAs. Additionally, it is possible to get data onto the watch with a special synchronization software [44].

The 1" display is monochromatic, the user interface basically the same as known from Palm PDAs. An infrared connection is possible for synchronizing the applications. To make this easier, an extra cradle is provided which has room for both the watch and the PDA. It is not possible to synchronize a desktop computer with the Wrist PDA. However, as [6] states, the Wrist PDA was canceled, but will be back in a new version in summer 2004.

2.1.5 Timex USB Datalink

The USB Datalink tries the balancing act between a sports watch and a normal PDA. This means it provides time keeping functions such as stop watches, alarms, different time zones etc., but also the possibility of saving personal schedules and telephone numbers [10].

The display of the Timex watch is not a pure LCD as most of the other projects uses them, it contains a seven segment display and two dot matrices (11x5 and 42x11 pixels).

Synchronization is done via USB, Microsoft Outlook can be used to transfer schedules to the watch. While the watch is worn on the wrist there is no synchronization possible since no wireless interfaces exist. Editing schedules directly on the watch is difficult [23].

2.1.6 Field Technology CxMP Ltd. Smart Watch

The Smart Watch manufactured by Field Technology CxMP Ltd. is definitely targeting a younger group. This watch shown in Figure 2.3 has a 256 color LCD with 72x64 pixels and comes in several different colors. The basic software configuration consists of scheduler, timer, stopwatch and even an image viewer and a melody player [56].

Figure 2.3: Field Technology CxMP Smart Watch. Image taken from http://www.smart-watches.com/images/product/images/ICW001/icw001_pict.gif. Copyright by Field Technology CxMP Ltd.

Different to the Timex watch or the Fossil Wrist Net, this watch can not be synchronized with Outlook, the schedules have to be entered with the software in the retail package. Photos or other pictures can be loaded onto the watch and displayed if they are cut down to the right size [40].

Synchronization is done with a serial cable and is therefore not possible while wearing the watch on the wrist.

2.2 Problems with previous projects

Several devices have been introduced, but all of them encounter some problems which should be avoided in future projects. In the following subsection, those problems will be shown and analyzed. Table 2.1 compares the features of all six devices.

2.2.1 Battery life

Biggest problem of all is the high battery requirement. The power supply of the Wrist Net watch's battery should last at least two days [27, p 18]. For a watch, this is a very short time. Even shorter is the lifetime of the IBM Linux Wristwatch's battery. Despite all efforts to save power it was not possible to extend the battery life beyond 2 hours [54], the complex operating system and graphical interface make their contribution here. Though it has to be mentioned that the IBM Linux Wristwatch is a technological proof-of-concept, not a commercially sold device. The Matsucom OnHand PC does have a strong battery drainage too and needs a new battery every two days—a fairly expensive task since it uses a coin battery. According to [29], the battery of the Wrist PDA should last at least four to five days, but it can be expected that this time drops down to a day or two in really extensive use, just as it is the case with the OnHand PC.

However, the time a complex watch can work without being recharged is limited. Effort has to be put into getting people used to recharging their watch regularly, just as they are used to recharging PDAs and mobile phones.

2.2.2 Processor power

IBM has put some effort into battery saving components: The processor used on the Linux Wristwatch runs on only a fourth of the possible speed to save

Vendor	Matsucom	IBM	Fossil
Device	OnHand PC	Linux Wristwatch	Wrist Net
Battery life	2 days	2 hours	min. 2 days
API	C (proprietary)	C (Linux)	n/a
Range	infrared	ca. 10 m	Radio based
Buttons	4	4 (touchscreen)	5
Other input	Joystick	n/a	n/a
Price	ca. USD 200 [17]	n/a	USD 179 [28]
Vendor	Fossil	Timex	Field Technology
Device	WristPDA	Datalink USB	Smart Watch
Battery life	4 to 5 days	2 years [3]	1 to 3 months
API	n/a	Assembler	n/a
Range	infrared	n/a	infrared
Buttons	3	3	5
Other input	rocker wheel, touchscreen	n/a	n/a
Price	n/a	ca. USD 90 [10]	USD 160 [40]

Table 2.1: Short overview about the features of the analyzed devices.

battery power [39]. On the contrary, this affects the speed of the applications running on the watch. During the test phase with the Matsucom OnHand PC problems occurred when sending a data stream over the serial connection to display images and video. Even using only 4800 baud the processor was not able to process the buffer fast enough, resulting in a buffer overflow on the serial interface and an operating system reset. The Wrist PDA suffers from its slow processor too. Finding a balance between low power and fast processors is difficult.

2.2.3 User interface

The design of the user interface has to have the battery issue in mind too—power can be saved by keeping the number of pixels to light up as low as possible. Many considerations about user interfaces on small displays have been made in [48]. In [39] several conventional watch faces are listed to show the relation of used pixels to the battery usage of the display. The manual for the Fossil Wrist Net states that "simple watch faces [...] take less power than elaborate watch faces, such as those that use animation" [27, p 18]. Looking at the OnHand PC shows that the application developers did not put the same efforts into a power saving user interface, animated screens and other power wasting effects are fairly common. The Smart Watch on the other hand switches the color display off after a certain timeout to prevent its battery from strong drainage.

Additionally, a huge reduction of battery life can be seen in all devices using backlit color displays. Monochromatic displays just light up pixels which should be black on the screen, relying on the reflection of the sunlight to be readable. Color displays have to actively emit light, making it readable even in darkness but resulting in an enormous battery drainage.

2.2.4 Connectivity

Another problem with the Matsucom OnHand PC is the weak connectivity. To get data onto the watch or vice versa, it needs to be connected via the serial dock or via infrared, which needs a direct line to the receiver. It is not possible to transmit data while the watch is worn normally on the wrist. The Wrist PDA and the Smart Watch only support infrared as well. The IBM Linux Wristwatch's connectivity is much better by using Bluetooth, though Bluetooth has the disadvantage of a small range. For most mobile purposes class 2 Bluetooth devices are used with a range up to only ten meters [2]. With a range this short it is, especially outside, difficult to have a permanent connection to the internet.

The Fossil Wrist Net has the longest range by using FM radio for data reception. The drawback of the radio solution though is that it is country specific. When the Wrist Net was introduced in January 2004 [16] the service was only available in the US. Additionally, the radio based solution does not allow a bidirectional connection, the Wrist Net therefore cannot send information to the sender. Another drawback is that a radio based solution can not be used for personalized information easily. The Datalink USB does not have any wireless connectivity.

2.2.5 Extensibility

As mentioned in Section 2.1.1, the Matsucom OnHand PC provides a C-API. The Datalink USB allows programming in assembler language and has quite a rich API as well. Both APIs are unique and require the programmer to learn a new interface for applications. The Wrist Net watch does not support any programming interface at all. It was not designed to work with other applications than the firmware, the application range therefore is limited to the available channels. The IBM prototype uses Linux as operating system and a programmer can then use the same interfaces used from desktop Linux programming, there is no need to learn a completely new API. This is the most preferable way to do this.

2.2.6 Interaction

The ways how to interact with the software are pretty similar on all watches. Both the IBM and the Matsucom watch provide four buttons for interaction, the Wrist Net watch five. The buttons of the IBM watch are sensitive parts of the touchscreen, whereas the other two use normal buttons on both sides of the watch case. Additionally, the Matsucom OnHand PC has a small joystick with four different states. This joystick can be used for text entry, similar to writing text on a game console. The Fossil Wrist PDA has a three direction rocker switch and three normal buttons and in addition the option to use the touchscreen with a stylus. Both the Timex Datalink USB and the Smart Watches only have normal buttons, three on the Datalink USB, five on the Smart Watch.

A good alternative would be speech input and speech recognition as described in [46] though with the current technology it is not possible to use this on a low power device.

Chapter 3

Concepts

This chapter gives an overview about the concepts of this project. First, Section 3.1 will explain the basic idea of this thesis, namely to use two different but tightly connected devices. Section 3.2 will discuss possible hardware for the watch. After that, methods how to switch applications on the watch are explained in Section 3.3. Finally, a system to switch the server devices at runtime is described in Section 3.4.

3.1 The Concept of the PersonalServer

To avoid some of the problems mentioned in Section 2.2 the concept of the the PersonalServer is used. This server is a software running on an additional physical device to offload most of the watch's tasks. The PersonalServer is responsible for all complex program logic, the watch only receives, decodes and displays pure image data. Connections to external data sources such as the internet, other computers or even household devices are always between the PersonalServer and the data source, not the watch device itself. The watch is only connected to the PersonalServer, even external input devices, as will be discussed in Section 3.2.3 connect to the PersonalServer instead of the watch. Figure 3.1 shows an illustration of this concept. From this separation the following advantages are expected:

- Less computing cycles on the watch and therefore a reduction of energy need while software is running.
- With the reduction of computation a simpler, more energy-saving and cheaper hardware on the watch.
- By offloading tasks to a more powerful device a faster response of complex applications.
- A better connectivity to the internet and other devices.
- A simpler interface for application programmers to use the watch as output device.
- A simpler way to install applications on the watch.

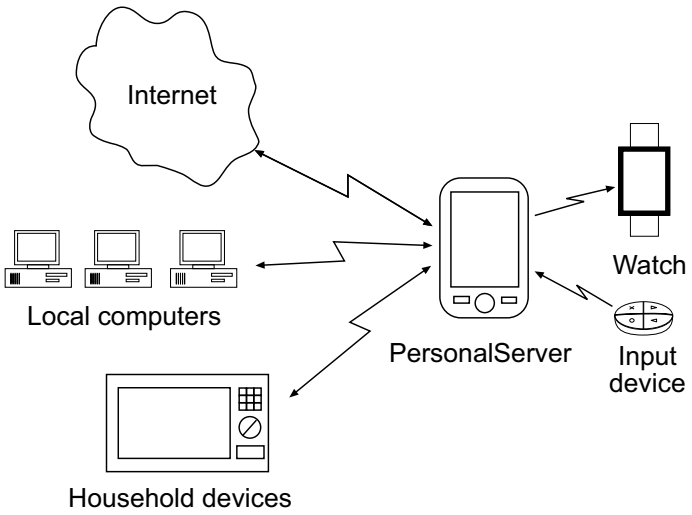


Figure 3.1: The separation concept of the PersonalServer.

In [62], a concept is described where a mobile screenless device makes use of brainless ubiquitous displays for displaying data. Roughly, the concept of the PersonalServer is quite similar to this one, except that the remote display is carried around all the time. The concept can also be compared to the concept used on SunRay terminals where the terminal only "processes user input and screen output" [21]. The terminal uses the X protocol to talk to the server, the actual processing is done by the X server running on the server machine. In this project, the PersonalServer prepares the image data for the watch and sends it over to the watch. This can be seen as simple implementation of the X architecture.

3.1.1 The PersonalServer Device

Since as much program logic as possible is offloaded from the watch to the PersonalServer, the watch is completely dependent on the second device and unusable while not connected to the server. Therefore, a physical device is needed which can connect to the watch easily, provides enough computational power and is small enough to be carried all the time. In an outside environment, this excludes a desktop computer as well as laptops since the latter cannot be used while walking, standing or driving in a car. Mobile phones would be possible since most users are already used to carrying them around. But they do not provide enough power to process complex applications and hardly ever have enough memory to store a vast amount of applications. Additionally, connectivity is difficult: Even though most newer phones do have technologies like Bluetooth it is only partly useable since the Bluetooth controller is usually part of the phone's operating system and mostly limited to a serial port emulation and sometimes proprietary implementations.

Another class of possible devices are PDAs. These are pretty common in the target group of business persons, have enough computing power and newer

Figure 3.2: The default screens of Microsoft PocketPC (left) and PalmSource PalmOS (right).

models support wireless technologies such as Bluetooth and WLAN. Currently, there are two main categories, PDAs running PalmSource's PalmOS and those using Microsoft PocketPC as operating system (see Figure 3.2). A big advantage of PalmOS is the simplicity of its GUI—maybe one reason why it still has the biggest market share of all PDA operating systems [7,18,34]. However, PalmOS has no multitasking ability which discards it as the PersonalServer since the PersonalServer software has to be running permanently on the device. With expectations that PocketPC will catch up to PalmOS in the next years [18,19] PocketPC is a good alternative. It has to be mentioned though that after problems occurred with PocketPC and its Bluetooth stack as well as the speed of Java in general (see Section 4.1) the decision was to use Linux as operating system on the PDA. Linux is comparable to PocketPC in its interface and functionality, the main drawbacks are that it is not officially supported by most PDA vendors and does not run on every PDA. Further reasons for Linux and why it was used as operating system for this project can be found in Section 4.2.1.

With the decision towards PocketPC a suitable device had to be selected. The HP iPAQ handheld computer supports Bluetooth and WLAN out of the box in its newer version and is already quite common in the target group. Also, two models of this PDA were available for prototyping in the research lab where this prototype was developed, which was the main reason for choosing this device. Since the operating system abstracts the hardware it was possible to implement the PersonalServer mostly hardware independent, thus making it possible to use the same software on a different device in the future. Figure 3.3 depicts how applications are insulated from the hardware on the PersonalServer device.

3.1.2 The Watch Device

As mentioned above, expectations are that with the separation into two devices less computations are executed on the watch. Since computation needs power, this results in less need for energy, especially since less powerful and more energy saving microcontrollers can be used. Additionally, most microcontrollers have power saving sleep modes, which can be then used far more often. The prototype used had a MSP430 with only 4 MHz (see Chapter 5).

This "parasitic computing" [47] allows it to do more complex tasks too. One of the project's goals was to allow video streaming on the watch. Even an Intel Pentium 60 only achieves about 9 fps when decoding MPEG streams [42],

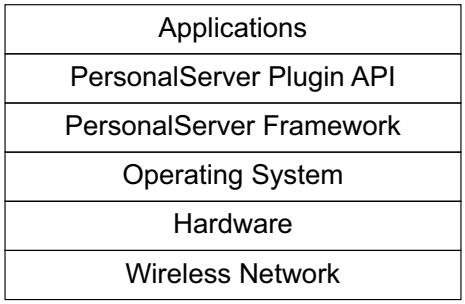


Figure 3.3: The PersonalServer system architecture. The application is using the plugin API which is tightly connected with the PersonalServer framework. Accessing hardware is done by the operating system on the device.

on an IBM Linux Wristwatch running at 18 MHz this would be completely senseless. By using two physical devices the PersonalServer decodes the video and then transmits only image data to the watch, which then only needs to light up the display. If a desktop computer is used as PersonalServer the available computation power will multiply.

As mentioned in Section 2.2, one of the problems is that it is very hard to connect to other networks from a wrist watch. Bluetooth only allows ranges up to 10 m on mobile devices and thus is only reliable inside where enough Bluetooth hotspots are provided. On the other hand, more and more cities provide WLAN hotspots in the city centers, train stations and even in restaurants like Mc Donald's [9], so the application domain becomes much bigger by using WLAN.

By using the PersonalServer concept, technologies like WLAN can be used indirectly. In Europe, the maximum power output for WLAN chipsets is 100 mW [37, p 52] and although most of the chips only use 30 - 50 mW [26] it would still be too much for a pure wrist device. Bluetooth on the other hand only needs 2.5 mW [31, p 21], so it is better to use this technology on small mobile devices. By splitting in two devices, Bluetooth can be used on the watch and WLAN on the PersonalServer. Since the PersonalServer is carried on the person the watch is easily within the maximum of 10 m. The PersonalServer software then routes data from the WLAN connection over the Bluetooth connection to the watch.

Using WLAN decreases the battery life of the PersonalServer (which is only 12 h in the case of an iPAQ [5]), but this does not have the same effect on the user since users are used to leaving the iPAQ in the cradle when it is not used.

3.1.3 The PersonalServer Framework

So far only the hardware of the PersonalServer was discussed, not so much the software. Finding a compelling application for the watch is hard. However, what makes the watch a special device is more the combination of applications, the easy development of new ones and therefore a widespread application domain. What has to be provided is the chance to create new applications with very low effort. Just providing one application would be wrong and the project's

goal—to replace a normal time-displaying wrist watch with the PersonalServer watch—could not be achieved. If just one application runs on the watch the target group does not have a reason why to use the watch in addition to the iPAQ. By providing applications in great variety the watch's basic goal—to have all necessary information always visible on the watch—gets closer. If the iPAQ never has to be used to get information unless to provide complex feedback such as text input, then the original goal of the project has been achieved.

It has to be considered though that even a set of applications is not useable if they do not work seamlessly and efficiently. If every application has to initialize the display and has to connect to the watch, development will be slowed down. What is needed is a library to make this tasks easier. The drawback though is, that a library can not control the data flow to the display. If two applications started up at one time, the transfers would interfere and the display would get inconsistent. So one solution to this problem is a daemon running in the background, paired with a library. Applications could use this library to draw onto a virtual watch display, pass the information on to the daemon which is responsible for multitasking duties. This attempt could easily be combined with existing applications. One could implement a plugin for i.e. the extensible mail reader Mozilla and use the daemon to show incoming mails on the watch. This is just one case where an existing application could be used to get data onto the watch without much programming effort. Still, many applications are small and may not be connected to existing applications. For those, initializing the display on the PersonalServer is already too complicated and can waste programming time.

The decision was to develop an application framework which provides all of the mentioned things. For each application two virtual displays are created, one for the local display on the PersonalServer and one for the remote display on the watch. Additionally, library calls are used to draw on these displays. The daemon mentioned above is then just one type of possible applications—listening on the network for image data to be sent to the watch. In other words, the framework running on the PersonalServer has to be a simplified operating system for the watch. An operating system provides access to the hardware and the possibility to run applications. In this case the hardware are the I/O devices on both the PersonalServer (the display, the touchscreen and the keyboard) and on the watch (the display). Simple calls have to be provided to access the display of the watch with drawing functions which can do graphic primitives and text. This is mandatory since an application should not have to care for which pixels to light up when drawing a line across the display. Also, it is ensured that only one application can send information to the watch at one time thus keeping the display content consistent. Figure 3.4 shows how a framework handles redraw attempts.

In the implementations, applications use the drawing functions of Java AWT and X. Both libraries are pretty simple but only provide a small amount of functionality. There is no 3D support on either of them like OpenGL does provide it. Although an iPAQ is capable of rendering complex 3D scenes as [12] shows, it did not seem necessary here. A 3D-API would be needed for games, but currently there is no chance for interaction on the watch itself so games are ruled out. The application domain definitely lies on the 2D interfaces so AWT and X can be considered good enough for most applications. Applications which need more complex interfaces (especially on the PersonalServer) still can use other

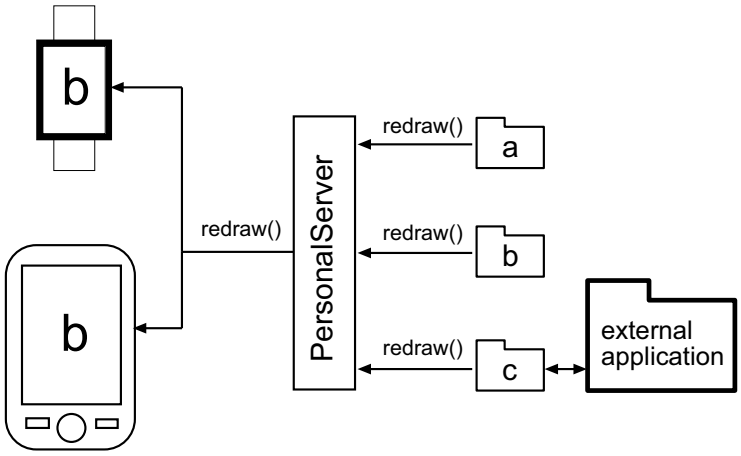


Figure 3.4: The PersonalServer framework. Applications (a, b, c) send redraw requests to the framework. Depending on the currently active application the screen is redrawn. External applications can use the framework via wrapper applications.

than the *java.awt.Graphics* class or GTK or Qt functions instead of X.

One important aspect of the framework was to make installing and executing new programs as simple as possible. This is already simplified by separating the watch and the PersonalServer. On the Matsucom OnHand PC for example it is necessary to put the watch into a cradle, then start up the Matsucom application on the computer and transfer the new applications to the device. By using the PersonalServer this is much easier, since the iPAQ usually spends most of the time in its cradle anyway to synchronize with the desktop computer and recharge its batteries. If the desktop computer is running Microsoft Windows with installed ActiveSync, the iPAQ is shown as another drive and it is easy to copy files to and from the iPAQ. If Linux is used on the iPAQ, file transfer can be done using the Networking File System (NFS) or Secure Copy (SCP). Even a Samba port is available on the iPAQ. All these transfer methods are very common under Linux and do not force the users to change their habits.

Ideally, no installation routines are necessary, applications are copied onto the iPAQ and available immediately. Deinstalling should work as simple as just deleting the application files. This is solved with a plugin system in both implementations (see Sections 4.1 and 4.3). Though it makes updating, installing and removing applications very simple this method has a drawback: The PersonalServer is completely separated from the applications and their logic, it does not know which application is responsible for which task. However, this can be solved by adding another auxiliary application which starts up the requested application on demand. This can be compared to the K Desktop Environment (KDE) under Linux where an additional application is needed to associate file types with the user defined program.

3.2 The Hardware of the Watch

A short overview is now given about the possible hardware of the watch itself. With the separation into two physical devices—the PersonalServer device and the watch—most of the tasks can be offloaded to the more powerful PersonalServer device, thus allowing to build the watch very simple. This chapter discusses which technologies are suitable for the watch and for what reasons. The real prototype, developed by the Hewlett Packard Labs in California, USA, differs from the ideal watch. For a detailed description of the prototype see Chapter 5.

3.2.1 The Display

The most important part of the watch is the display since this is the main interface to the user. Its size has to be big enough to render text and graphics but small enough to allow it to be worn on the wrist conveniently. Big wrist watches like Casio's G-Shock collection have round watch faces with diameters of around 5 cm, Round displays however, are very expensive and complicated to produce. A square display of the size 5x5 cm would look chunky and would be too big for an average wrist (see Figure 3.6). It would overlap a smaller arm on both sides and hurt when bending the wrist. Therefore, the display size has to be reduced, considering that other elements like batteries and user interfaces need to be packed on the wrist as well.

Computer screens, TVs and most mobile devices have a display proportion of 4:3 and standard LCDs are manufactured in the same proportion. A decision has to be made between the landscape format (long side on top) and the portrait format (short side on top). Each format has advantages: Using portrait format allows a bigger display since it is more convenient to have a long narrow watch than a short wide one. As Figure 3.6 shows, 4x3 cm are quite acceptable, whereas 3x4 cm (landscape) would be to big to be worn on the wrist. On the other hand "fewer lines with more characters were easier to read then fewer characters and more lines" [46, p 38], so a landscape format is preferable for readability. As Figure 3.5 shows, this effect gradually disappears with decreasing font size. Due to the size of the display and the software setup provided with the prototype the decision was to use portrait format.

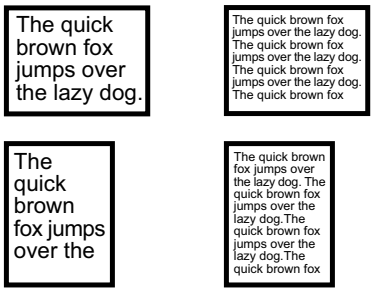
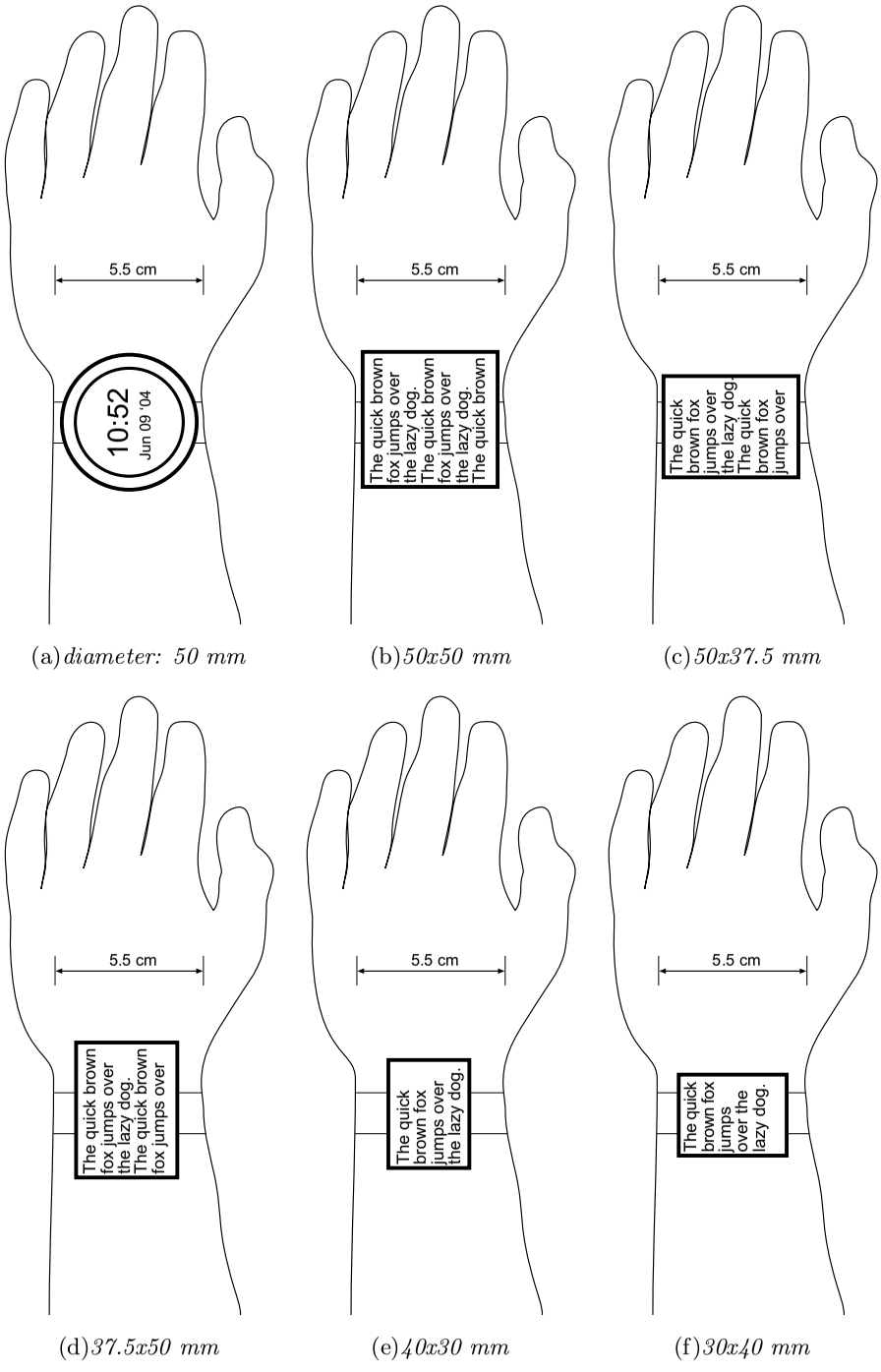

The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox
The quick brown fox jumps over
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.The quick brown fox jumps over the lazy dog.The quick brown fox

Figure 3.5: This example shows that the font size decreases the advantage of the landscape format in readability.


5.5 cm
10:52
Jun 09 '04
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over
The quick brown fox jumps over the lazy dog. The quick brown fox jumps over
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

(a) *diameter: 50 mm* (b) *50x50 mm* (c) *50x37.5 mm*

(d) *37.5x50 mm* (e) *40x30 mm* (f) *30x40 mm*

Figure 3.6: Different display sizes in relation to an average hand.

Resolution	Color depth	Data in Bytes	Transfer time in seconds	
			at 723.2 kBit/s	at 11 MBit/s
102x64	1 bit	816	0.0011	0.0001
102x64	8 bit	6528	0.0088	0.0006
102x64	16 bit	13056	0.0176	0.0011
102x64	24 bit	19584	0.0264	0.0015
160x120	1 bit	2400	0.0032	0.0002
160x120	8 bit	19200	0.0259	0.0017
160x120	16 bit	38400	0.0517	0.0033
160x120	24 bit	57600	0.0776	0.0050
320x240	1 bit	9600	0.0129	0.0008
320x240	8 bit	76800	0.1034	0.0067
320x240	16 bit	153600	0.2068	0.0133
320x240	24 bit	230400	0.3102	0.0200
640x480	1 bit	38400	0.0517	0.0033
640x480	8 bit	307200	0.4136	0.0266
640x480	16 bit	614400	0.8272	0.0532
640x480	24 bit	921600	1.2408	0.0799

Table 3.1: Data transmission times at different screen resolutions.

Another aspect of choosing a display is the display's resolution. A standard computer monitor has a resolution of 72 DPI, not enough for a mobile device with a display size of only a few square inches. The more DPI, the more information can be displayed, although there is a limit when the resolution becomes unreasonable. The Matsucom OnHand PC has a display size of only 102x64 pixels which makes it difficult to show a reasonable amount of text on the screen. Contrary, IBM uses a VGA display with 640x480 pixels which allows a larger amount of information. However, since each screen needs to be transferred from the PersonalServer to the watch the bigger the resolution is the longer one redraw attempt needs to be finished (see Table 3.1).

For a VGA display, the amount of data is eleven times as big as the data for the Matsucom display, if the display is not monochrome but with color the amount is multiplied. As Table 3.1 shows that at a theoretical data transfer rate of 723.2 kbit/s, the data transfer rate of Bluetooth in asymmetric mode [31, p 41], a VGA true color (24 bit) display would need more than a second just for transferring the data, not including the time for processing and displaying the data. However, by reducing the color depth to 8 bits per pixel and the size of the screen to 120x160 it is possible to get down to roughly 32 ms. This value can be decreased by applying various compression algorithms on the data before transferring it.

3.2.2 Connectivity

Since the decision was to make the watch as simple as possible, good connectivity is needed to transfer the screen contents from the PersonalServer to the watch. Currently there are several standards for wireless connections. This section gives an overview about the most common ones and which one is the preferable for the watch.

The following five technologies are common in mobile wireless communication:

- Infrared is one of the older standards and well known since mobile phones came up. It supports up to 4 Mbit/s but only 9.6 kB/s are mandatory [38, p 7]. Most phones only support the mandatory 9.6 kB/s, but Table 3.1 shows that this data rate is not enough for transferring images. The biggest drawback however is that an infrared transmission always requires a visible line between sender and receiver. Since the plan was to have the PersonalServer somewhere in one's pocket an infrared connection does not suit the requirements.

- FM radio waves can be used for transmitting binary data as the Fossil Wrist Net shows. The big advantage here is that radio waves are useable practically anywhere, there is no maximum range. However, this only works as long as there is a common information for all users, not the personalized information the PersonalServer provides. Since the infrastructure for sending radio waves is different a license for transmitting data within the radio frequency band is needed in most countries. Even more likely a contract with a public radio station needs to be made use their transmission hardware. This again results in enormous costs if personalized information should be provided. Another problem here is the security issue with personalized but broadcasted information.

- GSM/GPRS/UMTS are common for telephony today and for data transmission outside of cities. WAP was the first mobile data application over GSM. Nowadays, modern phones often provide full HTML browsers and even Java. More and more PDAs support GSM as well and can be used for telephony. However, this technology has its drawbacks: Only a few phone providers support sockets over GSM/GPRS, most providers only support the stateless HTTP. This would not be so much of a problem, the PersonalServer could be programmed in a way to support HTTP. The bigger problem is that the providers charge either for connection time or for the transferred data. A permanent connection would therefore be expensive and unaffordable.

- Bluetooth is a standard which is used more and more often in modern mobile devices. Some years ago most mobile phones had infrared connectivity, now they have a Bluetooth chip instead. Bluetooth is wireless standard working with radio waves in a frequency band around 2.4 GHz so no direct visible line is necessary. The maximum transfer rate in asymmetric mode is 723.2 kbit/s in one and 57.6 kbit/s in the other direction [31]. The maximum range depends on the used device class, class two, the most common one in mobile devices, has a range up to 10 m. Bluetooth was specifically designed for ultramobile devices and supports some very useful features other standards lack. The network topology is divided into so-called piconets with eight devices each, though communication over the bounds of one piconets are possible by bridging. Bluetooth supports a vast number of protocols including TCP/UDP and IP for Personal Area Networks (PAN).

Technology	Advantages	Drawbacks
Bluetooth	Cheap Very common Low power needs	Low range
WLAN	Long range high data rate	High energy need
IrDA	Cheap Very common	Low range Visible connection necessary Low data rate
FM	Network dependent range	Infrastructure expensive no personal configuration no feedback channel
GSM	Very common Network dependent range Existing infrastructure	Low data rate Expensive
GPRS	Very common Network dependent range Existing infrastructure	Low data rate Expensive
UMTS	Network dependent range Existing infrastructure	Expensive Infrastructure not yet fully available

Table 3.2: Feature comparison of wireless technologies.

- WLAN - Wireless Local Area Network is a fairly new standard and supports bit rates up to 11 Mbit/s or 54 Mbit/s, much higher than the Bluetooth or infrared data transfer rate. The used frequency band is in the free 2.4 GHz ISM band, the same Bluetooth uses. WLAN is becoming very common in newer PDAs and laptops, but still uncommon in ultramobile devices such as phones. This is mainly because of the high energy need of up to 100 mW, which makes it unusable for the watch too.

As noted before, IrDA, FM and GSM/GPRS/UMTS cannot be used for the project due to several reasons. From the two remaining alternatives, Bluetooth and WLAN, Bluetooth seems to fit the requirements better for a mobile device. The chipsets itself are cheap and connectivity is easy with most modern devices. WLAN has advantages in speed, range and the availability of public hotspots but needs too much power for a ultramobile device.

3.2.3 User Interaction

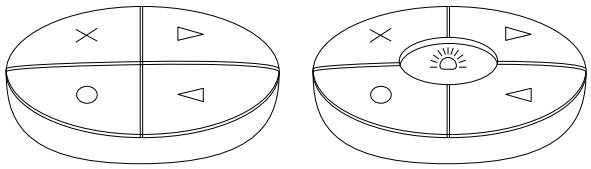
Another question coming up when thinking about the ideal hardware is which interaction devices are necessary to give a useable and easy interface. Nearly all digital watches provide buttons, mostly four. On most mobile phones there is a twelve button keyboard and additionally four to six buttons for going through menu options. Since the keyboard is only used for dialing and writing (one exception are games, but in most games the keyboard is reduced to five keys) it is possible to control a—compared to a watch—rather complex menu with four

to six buttons as well. So the watch should be fitted fine with four or maybe six buttons.

Another approach is the so-called jog wheel, as used on i.e. the Sony Ericsson CMD-Z5 mobile phone (see Figure 3.7). A jog wheel is a small wheel, usually used for going through menu options, which works as a button as well. So three different interactions (up, down, press) are packed on closely the same room a button would need. A common problem with jog-wheels though is that they are sometimes very hard to use. Some of today's phones have a little joystick (i.e. Sony Ericsson T68i shown in Figure 3.7), which supports four or eight different directions. Depending on its size, a joystick might be easier to use than a jog wheel, but then it needs more place. One important point about joysticks is that they have to point upwards, whereas buttons or jogwheels can be on the side of the watch as well.

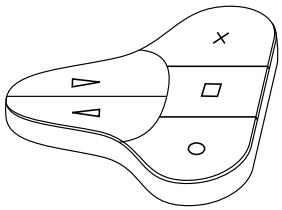
The whole issue of wasting space can be solved by using a different concept, namely to separate the input device completely from the watch. Two devices are available then—the watch, only used for displaying data and the input device, consisting only of buttons, a microcontroller and a Bluetooth chip. Although two devices have to be carried around then there are advantages of this concept: Using the buttons on the watch the user needs both hands since they cannot be reached with the hand where the watch is worn. The separate input device can be used with the same hand, thus leaving the other one to carry a suitcase or to hold on to a railing in public transports. And since the device is completely separated from the watch it is interchangeable. Like gamepads on game consoles the user can buy better, smaller or more complex ones to fit his needs. It has to be kept in mind though that then the PersonalServer has to be adjusted to this concept and kept more flexible. A situation could occur that the users might not have any input device at all or that they have a very complex one to browse through menus and applications. Drawings of examples for this separate input devices can be seen in Figure 3.8.

Figure 3.7: Sony Ericsson CMD Z5 with jog wheel and Sony Ericsson T68i with joystick.

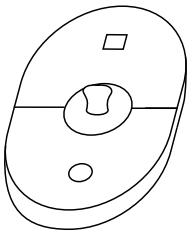


(a) *4 button device*

(b) *4 button device with optical mouse*



(c) *5 button device*



(d) *2 button device with joystick*

Figure 3.8: Example sketches of separated input device

3.3 Switching applications on the Watch

As mentioned in Section 3.1.3, a framework is provided for the applications to allow easy access to the watch's hardware. Most applications are just optimized for one task so it has to be expected that more than one application is running on the watch at every time. Due to the display size, only one application can access the display at once[1]. Therefore, it is necessary to provide a method how to switch applications at runtime.

3.3.1 Task Manager

A well-known approach is to implement a task manager like the one used in Microsoft Windows, KDE, Gnome and other window managers. By pressing the key combination Alt+Tab the user can switch between applications. A similar concept runs on both implementations: The user can view a list of the running applications and then bring one in the foreground. However, keyboard combinations are rather complicated on the on screen keyboard of the iPAQ, so the Alt+Tab approach is not preferable. On the other hand, there is a touchscreen, which makes it easier to select an application as soon as there is a visible list—a click on the application instead of cycling through saves time. There are buttons on the iPAQ too, but those have their own standard configuration which should not be changed.

So the buttons have to be placed somewhere on the GUI of the Person-

[1] In the actual implementation two applications access the display at once. One of them is the clock application, which has a small fixed area on the display which is not accessible by the other applications.

alServer. Doing that means losing some place on the already very small display but it seems inevitable. It is hard to emulate right clicks, which eliminates context menus.

The majority of the users are familiar with the concept of a taskbar, which provides fast access to open applications. In this case an implementation in a similar way to the MDI interface of the Norwegian web browser Opera is needed. This program has its own taskbar within the GUI and does not use the operating system's taskbar as Microsoft's Internet Explorer does. On the iPAQ however this concept fails because of the small display resolution. The width of the Display (240 pixels) is too small to show a reasonable number of applications, so in the taskmanager's implementation all running applications are displayed in a scrollable list on the whole display. This eases development too since the same plugin system the other applications use can be used, including all benefits: short implementation time and the chance for the user to change the taskmanager against a different one. However, since the taskmanager is one of the most important applications it is not recommended to remove it completely.

3.3.2 Interactivity on the Watch

More difficult is to provide a method to switch applications on the watch without using the PersonalServer device. Possible devices for interactions were discussed in Section 3.2.3 and to realize the task manager on the watch at least one button is needed. By pressing this button the task manager is started and shows the applications. While holding the button the task manager cycles through the applications and on release the currently selected applications appears on top. Early cellphones used this method to write SMS. However, the biggest problem is that the cycle delay must not be too fast to allow a proper selection. This on the other side raises the average time to select an application beyond the acceptable maximum when many applications are open.

A solution using two or three buttons is preferable here: one button to start the task manager and to confirm the selection and one or two buttons to cycle through the list. This concept was implemented on the Matsucom OnHand PC while it was used as a replacement for the watch and it was acceptable handy.

However, the best approach for flexible interactivity is a separated input device as mentioned in Section 3.2.3.

3.3.3 Smart Application Switching

If no devices at all are available for interaction, the iPAQ is needed to switch applications on the watch. To avoid that, a solution to switch applications automatically is preferable. The PersonalServer has to know which application has a higher priority than all others in order to be drawn on the display of the watch. Four categories of applications can be distinguished:

- Applications which use the display for a very long time but do not have important information at one specific point. An example would be a video stream from a baby monitor. While the information provided over a long time is relevant, the information at one specific second is not especially important. Therefore the video stream can be interrupted for some seconds to display other information.

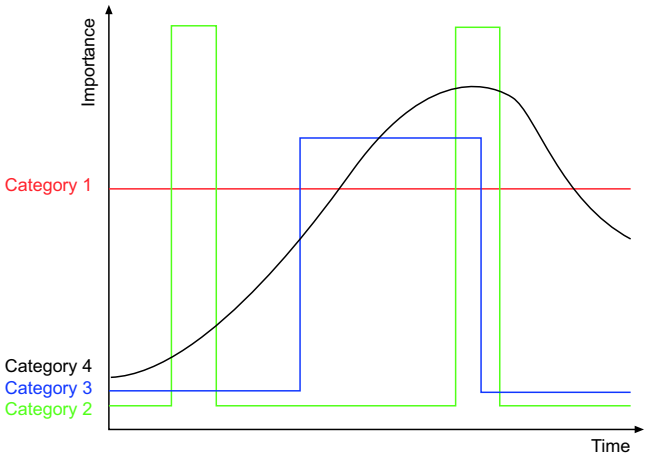


Figure 3.9: Example of how different applications can have different priorities over time.

- The second category are applications which provide highly important information but only for a very short time. Email notifiers are in this class of applications. Within five seconds after an email has arrived this information is very valuable, until the next email arrives there is no information worth displaying.
- Another category of applications have information limited for a certain amount of time. A video stream sent from a door bell has important information over a time of about 30 seconds until the user opens the door, but not longer.
- The last category are those whose amount of valuable information changes over time. A countdown on a timer application for example is more important when it is close to the end.

Figure 3.9 shows the importance of the information of the four application categories in a diagram. Whereas category one (red) stays the same, categories two (green) and three (blue) have difference importance over time. A category four application (black) might not be deterministic at all.

It can be expected that usually one of the long lasting applications is running, several more of the second type. An example scenario would be a user sitting on the couch, watching TV and watching the baby sleeping over the baby monitor. From time to time a notification about incoming emails is displayed, when the TV series end the watch displays information about the next series starting on this channel. These short information is very helpful but only necessary for a short time, thus interrupting the video stream but not rendering it useless. Implementing a system for a scenario like this is fairly easy.

It gets more complicated if an application of the fourth type is running. If the user puts a pack of popcorn in the microwave, a countdown is displayed on the watch. This cause problems now. The amount of information is difficult to estimate. The shorter the time is, the more information it provides. It is

very important to know whether it is 20 or 10 seconds to the end, but it is less important to know whether it is 5:10 minutes or 5:20 minutes to the end. To display the most important application the PersonalServer has to evaluate the amount of information of every running application.

The same problem occurs when using an application of the third class. A video streaming from the door bell interrupting the baby monitor for 30 seconds is acceptable, but the same video stream would not be important enough to interrupt a live share prices display during a meeting. Since the information depends much on the context of the user, it is not possible to let the PersonalServer solely decide over the applications' priorities.

A system is needed where the user can define the priority of an application, depending on the current context. The user should be able to define a maximum priority for each application, the application itself can then ask the PersonalServer to be drawn with a certain priority up to this user defined one. The PersonalServer compares this priority with the priority of the application currently on the screen and then displays the application with the higher priority. With this method the user can set the priority for the share prices application very high, thus avoiding to get interrupted by an incoming email.

3.4 Switching PersonalServer

A common situation is that the PersonalServer device becomes obsolete under certain circumstances and the PersonalServer framework has to be transferred to a different device on the fly. This could happen when the users come home and enter the Bluetooth network of their own house, or as well during the day, when they are covered by a Bluetooth network in the company as sketched in Figure 3.10. During this time there is a need for different applications than while being outside.

For a company the possibility of equipping employees with the watch and controlling the applications on the watch from a central server might look attractive. However, since a watch is a very personal device, much more direct in communication than a PDA, it might be uncomfortable for the users if their watch is "taken over" by the company as soon as they enter the building since—

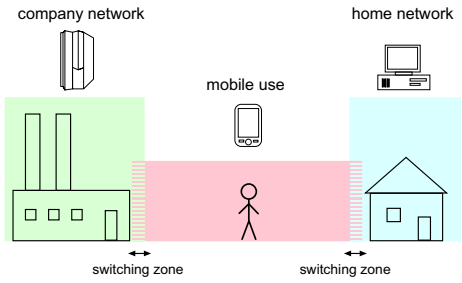


Figure 3.10: Switching PersonalServers. In mobile use the Watch is covered by the iPAQ as PersonalServer device. In the company or at home the PersonalServer moves to a different device, such as a central server or a desktop computer.

different to their setup at home—they would not have control over the applications running on the watch any more. A PDA can be easily ignored, with a watch worn on the wrist this is much harder.

Additionally, problems come up if the number of devices or the network itself becomes to big. As mentioned in [64] the more complex the network gets the greater the costs in routing and network overhead. Also, authentication, recommended for a personal device like a watch, gets very complex.

The scenario is still interesting: While standing in the tramway the watch displays a news ticker and the task list for the following day. As soon as the users enter their house the watches switch to the local PersonalServer and load new applications. A message could appear, informing that the rest of the family is at the neighbors' house, afterwards the current TV program is displayed.

For this scenario it is necessary to implement both a method to recognize other servers and to transfer data to and from them.

3.4.1 Synchronizing Servers

To transfer data the alternative server has to be known to the current PersonalServer. This can be achieved by either broadcasting to a full network or by trying direct connects to certain IP addresses.

The first approach can be useful to detect unknown servers but implies security issues such as the tempted migration to a non-authorized server. This could happen when the user visits a friend's home and the PersonalServer tries to shift to the PersonalServer there.

A slightly safer approach is to provide a list of authorized IP addresses. The PersonalServer then tries in certain timeslots to connect to one of those addresses. If an alternative server is within reach, the migration can start. For security reasons this has to happen with the authentication of the user or with another authentication method such as a private/public key authentication [32]. However, the average user will not have more than one or two alternative servers, maintaining the IP addresses is not too time-consuming. One advantage of using Bluetooth is that this technology has built-in security protocols as described in [63].

In the ideal case the tool for shifting servers is an application matching into the plugin system instead of being built into the PersonalServer. This eases development and gives the chance to uninstall it in case it is not needed at all. Since every installed application is a possible security leak, this way is preferred over the built-in way. The application then listens on a specific port and tries to connect to a remote server at regular intervals. If one server responds, the synchronization (see Section 3.4.2) starts after a successful authentication.

A different problem occurs when an existing connection between two PersonalServers is lost as can happen when the user with the watch leaves the range of one PersonalServer. In this case it is not possible to shift the data to the new server, since the connection is already lost. Therefore, synchronization is necessary between the two servers while the connection is still active. Additionally, a system to detect the connection loss is needed. Heartbeat [53], as used in cluster solutions, is one way of supervising the state of distributed systems. In Heartbeat's active/passive mode the alternative system starts its services when the main system goes down. The synchronization of the data is not done by Heartbeat itself. In the case of a PersonalServer it is hard to determine which

server is the main system connected to the watch, as Heartbeat is not designed for mobile devices. In the current implementations of the PersonalServer there is no bidirectional connection to the watch, it is therefore impossible for the PersonalServer to find out if it is connected to the watch. This reduces the benefit of the server shifting approach since situations could occur, where the watch does not have a connection to the real server any more.

The solution for this is to start the shift manually. The connection attempts to alternative servers are still automated, the shift itself has to be invoked by the user. By implementing a bidirectional connection the servers could be priorized. The PersonalServer at the company would have higher priority than the one on the iPAQ thus resulting in a server shift as soon as the iPAQ has a connection to the company server. The iPAQ then stops transmitting data to the watch. As soon as the connection between iPAQ and company server is lost, the iPAQ tries to establish a connection to the watch and then resumes transmitting data to the watch.

3.4.2 Synchronizing Data

On synchronizing applications there is a need to differ between transferable and non-transferable applications. The decision which application is transferable depends on both the current system configuration and on the user. The latter must be able to deny the transfer of certain applications, as a company would avoid transferring confidential data to the PersonalServer of the user. Also, the system configuration of the alternative server might be different, as not all applications might be installed on both servers. In this case Java would be a big advantage since it uses platform independent byte code. So the application could automatically be transferred and continued on the alternative server. By using a native implementation in C, C++ or any other compiled language this is only possible if both PersonalServers use the same processor type, which is rather unlikely if the two servers are one mobile and one stationary device.

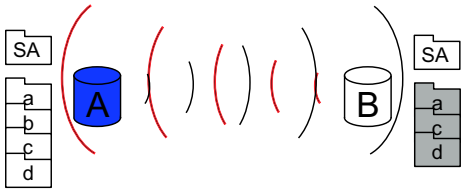
To synchronize the applications existing on both servers the synchronization application has to provide a list of the transferable programs on both the sender and the receiver side. By comparing this list the decision is made which applications have to be started on each server to keep the data synchronized. As mentioned in Section 3.4.1 it is very likely that the connection between two PersonalServers is interrupted at any time. So there is a need for keeping both servers synchronized while the connection is still active to avoid data inconsistency. This principle is similar to the one used in clustered systems with the Distributed Replicable Block Device (DRBD) [51]. The approach has to be different though. Other than the DRBD implementations there is not one specific device where data is written to but merely a whole set of applications, each of them with its own set of data. So there are two ways of transmitting the data to the alternative server:

- The applications are notified when and on which address a different server is available and then have to establish the connection to this server and synchronize the data by themselves. The big drawbacks here are the vast amount of open connections between the servers, which can be a problem if there is a firewall between the servers. In addition, applications have to implement the network code by themselves, thus making application

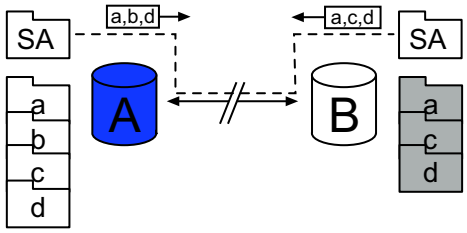
development more complicated.

- The applications are pushed by a synchronization application to provide exportable data. Consequently, the synchronization application sends the data to the second PersonalServer. On the other side this data is received and then presented to the matching application. Figure 3.11 shows this approach, which is slower than the one mentioned before but has certain advantages:
  - The synchronization application opens only one network connection to the remote server.
  - Applications get more lightweight since they do not need networking code for synchronization.
  - Applications do not have to care for half-transmitted data. Either data is transmitted as a whole or not at all. Error handling can be omitted and the developing time is reduced to a minimum.

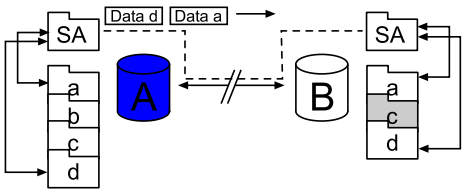
Due to the simplicity for application developers the second approach has to be preferred. However, as Heartbeat and DRBD show, developing a protocol for monitoring connections and synchronizing data are complex processes and goes beyond the scope of this thesis.



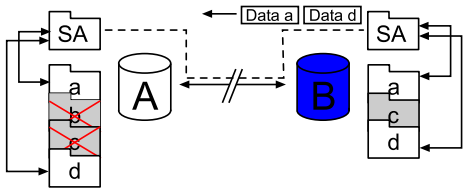
(a) Both synchronizing applications (SA) on server A and B are trying to establish a connection. A is the currently active server (blue).



(b) The two SAs are connected. A sends a list of synchronizable applications, B answers with a list of available applications. Currently, no application is running on B (gray).



(c) B starts Application a and b, A sends the data of both a and b to the SA on B. Application d cannot be started since it is only installed on A.



(d) B is now the active server. The applications b and c on A are stopped, a and d on B send their data to A.

Figure 3.11: Smart synchronization.

Chapter 4

The Implementation

The concepts described in Chapter 3 were realized in two different implementations:

- The first implementation was fully in Java, using Microsoft PocketPC as operating system on the iPAQ. Problems occurred with this implementation, mainly with the execution speed of Java (see Section 4.1.5), so a second PersonalServer was developed.
- The second implementation was in C, using Linux as operating system on the iPAQ.

Both systems are using a plugin system as previously described in Section 3.1.3. On the following pages, the implementations are described in detail.

4.1 The Java Implementation

This section gives an overview about the framework implemented in the programming language Java. At the time of this implementation the watch prototype was not yet available, so the display was emulated using the Matsucom OnHand PC.

4.1.1 Why Java?

Java is a high level object oriented programming language which does not run native but instead employs platform independent "bytecode" as intermediate language [1]. So what is needed to use Java on a certain platform is a compatible Java Virtual Machine (JVM) to execute this bytecode. There are several JVMs available for the iPAQ, one—the Jeode VM—is part of the iPAQ's retail package. Other JVMs were tested, such as the NSICOM CrEme, Ewe VM and SuperWaba VM, whereas NSICOM CrEme was from a personal view the best and fastest, but not for free. The decision was then to use the Jeode VM, as this one is surely available on every iPAQ and does not require special setup skills.

The bytecode is one of Java's greatest advantages. Java's "write once—run anywhere" principle [50] eases development on different platforms and is a big advantage if a different device is considered as PersonalServer device in

the future. Furthermore, this allows the transfer of running applications over a network to a different device, as mentioned in Section 3.4.2.

Additionally, Java is a high level programming language, resulting in a shorter development time, a big advantage when it comes to prototyping. In fact, this prototype was developed within only a few weeks.

4.1.2 The Plugin Framework in Java

A main aspect of the framework is to ease loading new applications at runtime. The interface[1] Java provides here is the so-called Reflection API. "With the reflection API you can: [...] Create an instance of a class whose name is not known until runtime." [24] The Reflection API provides instances of *Objects*, the parent class of all instances. To make use of those then, all applications used within the PersonalServer framework have to have a common Java Interface or an abstract parent class.

The decision was to use an abstract class as common interface. Java does not allow multiple inheritance, only multiple Java Interfaces. Choosing an abstract class as parent class prohibits developers to use other application's classes. This is the case if an application should work closely together with external applications. Though there are several reasons for the decision to use an abstract class:

- Java Interfaces only allow the declaration of methods and the definition of constant values. To use applications efficiently, several values have to be set by the PersonalServer at runtime, such as the dimensions and the *Graphics* objects of the displays.
- Several methods have to be implemented by the template to ensure a correct initialization of the application. This cannot be done within a Java Interface.
- A Java Interface forces the user to implement all methods in the implementing class. Simple applications might not need all methods provided by a parent Java Interface. Forcing developers to implement those methods although not needed is annoying for them. An abstract class allows the empty implementation of those methods, ensuring that developers only need to override those required by their specific application. The same concept is used by the *Adapters* for the *java.awt Listeners*.

For deploying applications Java provides the possibility of packing applications in Java Archives (JAR), which are basically zip files [11]. The Java Runtime Environments (J2RE) in the versions 1.2 and newer contain the class *URLClassLoader* to support the possibility to load applications at runtime from JAR files. So JAR files are a good way for deploying PersonalServer applications. However, the JeodeVM only supports the Personal Java Application Environment specification, which only requires full Java 1.1.8 compatibility in the *java.net* package [58] and does not contain the *URLClassLoader*. This functionality had to be implemented manually.

[1] Interface is a ambiguous word when writing about Java. In the following "interface" will be used to describe a interface between two software components, whereas "Java Interface" will be used to describe "a collection of methods and constant values" [8] which is specified with the *interface* keyword. For a detailed description about Java Interfaces see [25].

4.1.3 The Application Template

To allow the PersonalServer to access the applications, a common interface is needed. Therefore, a template is provided to define common properties of all applications. In the Java implementation this is the abstract class *Application*. The following key properties are provided:

```
public abstract class Application extends Thread
{
  protected static int rWidth;
  protected static int rHeight;
  protected static int lWidth;
  protected static int lHeight;

  protected String name = "Untitled App";

  protected Graphics lDisplay;
  protected Graphics rDisplay;
  [...]
```

The first four listed variables define the displays' dimensions. Each application has two virtual displays: a local one, representing the display on the iPAQ, and a remote one, which represents the LCD of the watch. *lDisplay* and *rDisplay* are the *java.awt.Graphics* objects which can be used for drawing.

Additionally, some basic functions are provided to allow interaction with the PersonalServer:

```
  public final void draw(int drawMask) { [...] }
  public final void draw() { [...] }
  public final void drawRawDisplay(int[] display) { [...] }

  public final InputDevice[] getDevices() { [...] }
  [...]
```

The two *draw()* methods allow the application to tell the PersonalServer that it needs to be redrawn. The PersonalServer then checks if this application is on top and therefore allowed to be redrawn. Then the pixels of the *java.awt.Graphics* object are fetched, converted into a readable format for the watch and sent. The method *drawRawDisplay()* allows an application to bypass the complex image grabbing algorithm and and provide binary data directly. This is helpful for applications with a high data volume (i.e. video streaming).

The method *getDevices()* provides a set of *InputDevices* to the applications. This is compliant with the concepts discussed in Section 3.2.3 on page 19. Those instances represent the input devices connected to the PersonalServer and allow it to make use of them. Different input devices can be used on the PersonalServer, each with its own set of buttons. Those input devices have to register at the PersonalServer during runtime and can then be used by applications.

Finally, *Application* implements four methods to allow the PersonalServer to interact with the application:

```
public void init() {}
public void close() {}
```

```
public void activated() {}
public void click(int x, int y) {}
```

The methods *init()* and *close()* are called as soon as the application starts or finishes and can be used for initializing or closing down network connections or an additional graphical user interface. If an application changes from being a background application to the one shown on the displays the method *activated()* is invoked to notify the application. Finally, the method *click()* is used to signal a mouseclick on a certain position.

Furthermore, as listed in the first code segment, *Application* uses *Thread* as parent class. Consequently, each application is running in its own thread, started by the PersonalServer. This allows an application to use blocking network code without interfering with the PersonalServer.

4.1.4 The Java PersonalServer

As Figure 4.1 shows, the PersonalServer consists of six main classes. The class *Application* was already described in the previous section and represents all loaded applications in this diagram. *PersonalServer* is the program entry class and the core module connecting the other modules. *PersonalGUI* and *RemoteDisplay* are wrappers for the displays.

The local display (the one on the PersonalServer device itself) is represented by *PersonalGUI*, which is responsible for initializing the AWT library and for fetching mouse clicks on the local GUI. Additionally it starts up the menu bar at the bottom of the display. All events happening on the local GUI are passed on to the PersonalServer, which then invokes the matching method of the currently active application.

RemoteDisplay is the representation of the watch's LCD. It connects to the watch using a UDP connection and does connection control tasks. Furthermore, the conversion of the pure pixel data into packets the watch can understand is done here as well. This packet format will be described in Section 4.4.

InputDeviceConnector is the control point for connected input devices. It is responsible for managing both the input devices themselves as well as the listeners on the input devices. By managing those in a separate class *PersonalServer* itself becomes simpler and easier to maintain. The *InputDeviceConnector* is listening on a special port for incoming messages. Those have to be compatible to a simple protocol, the so-called Device Data Transfer Protocol (DDTP). An input device has to support this protocol to send input data to the Person-

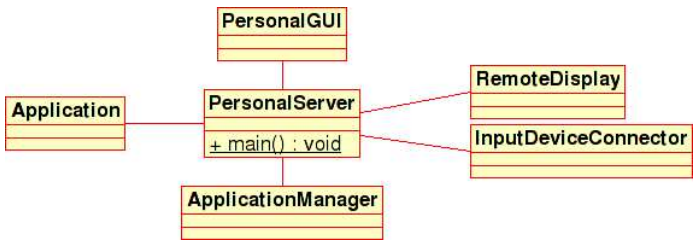


Figure 4.1: A simplified diagram of the PersonalServer architecture.

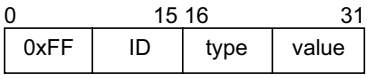


Figure 4.2: Structure of a DDTP packet. The first byte is used as a recognition sequence. The second byte specifies the ID of the input device, byte three the type of action. Byte four defines the value of the certain action.

alServer. The first arriving message is used to register the input device on the PersonalServer. Figure 4.2 shows the packet structure of a DDTP packet.

Finally, *PersonalServer* owns an instance of the *ApplicationManager*. Different to the implementation in C this task manager cannot be removed or changed, it is deep-seated in the framework. This is the case with *ApplicationStarter* as well, the reason for that is that applications are very restricted when it comes to accessing the parent PersonalServer, whereas the task manager and the starter both need extended access to the PersonalServer to manage the application lists.

Generally, it has to be said that the Java implementation was less modularized than the C implementation. Since the Java prototype was the first one several design errors were noticed and those tasks were done differently in C.

When the PersonalServer starts up an application, the *init()* method is invoked. After that, a new thread is started for this single application. Finally, *activated()* is called to notify the application that it should redraw the screen. By letting the applications run in their own threads there is no work for the PersonalServer unless an applications requests a redraw.

An application can call *draw()* at any time to redraw the screens, but the data is only transferred to the watch if the application is in the foreground.

4.1.5 Problems with Java

Different problems occurred with the Java prototype. One was, that the JeodeVM only supports the Personal Java Application Environment standard, which is only a subset of today's common J2SE 1.4. Many things done in the standard Java libraries in the JRE 1.4 had to be implemented by hand, as for example string handling.

The main problem was that although the time for developing a prototype is very short when using Java, the execution time is very bad. As explained in [41], this problem is mostly because of the GUI toolkits. However, the GUI cannot be avoided on the PersonalServer. Furthermore, reading directories and extracting JAR files needed a long time. If more than ten applications were installed, browsing through the different JAR files to get the necessary information needed over ten seconds.

A flaw in the implementation was that on each redraw the whole watch display was transferred. The update algorithm only transferring differences between the current pattern and the pattern already on the watch was then introduced in the C implementation.

Another problem which occurred when the PersonalServer was tested on the iPAQ was that the program's time to respond increased dramatically if the Bluetooth port on the iPAQ was receiving data. When searching for the error

Figure 4.3: Familiar Linux 0.7.2 running on an HP iPAQ 3870.

it was noticed that the same problem occurs when no Java program is running on the iPAQ. The startup time of the Internet Explorer for example multiplied up to several seconds when the Bluetooth port was flooded with video data. However, it remained unclear if the reason for this was a hardware error on this specific device or a bad implementation of the PocketPC's Bluetooth stack.

4.2 Linux on the iPAQ

Due to the speed problems with the Java implementation a reimplementation in a faster language was considered. The final decision was C, since this language runs fast and compilers are available for most operating systems. In addition to this switch in the programming language, a switch to using Linux as operating system on the iPAQ was considered. One of the reasons for this was the project Tinmith, a "software architecture for 3D mixed reality applications" and "optimized to develop mobile augmented reality and other interactive 3D applications on portable platforms with limited resources" [49]. Currently, a port of Tinmith to the iPAQ is considered and expectations were that the knowledge coming from this project can be used in porting Tinmith. The following sections will describe benefits of Linux and how Linux runs on the iPAQ. Figure 4.3 shows the main interface of the Linux installation.

4.2.1 Reasons for Linux

Using Linux has several advantages, one of it is that (although not preinstalled with Linux out of the box) more and more PDAs support Linux, thanks to several projects such as http://www.handhelds.org. In addition, Linux runs on a great variety of platforms, from desktop computers up to the IBM Wrist Watch. An small list of supported platforms can be found in [13]. This results in a limited form of platform independence. A program compiled to run under Linux on a specific platform can be recompiled on a different platform and expectations are high that it does work as well as it does on the original platform. For

developers, this is a great benefit, since programs can be tested and debugged on a desktop computer and then compiled and copied to the other platform.

The biggest advantage of Linux is the OpenSource concept. Both the Linux kernel and most libraries are published under the terms of the GNU General Public License[2]. Therefore, most development tools are free software and so is the compiler as well. For PocketPC, it is harder to find free compiler. Additionally, this license allows to view the sourcecode of libraries and tools, resulting in a better overall understanding.

Finally, from a personal point of view, programming under Linux is easier and more convenient, especially because of a great number of helpful tools.

A big disadvantage however is the missing hardware support. Often hardware vendors do not publish specifications, thus making it difficult to write drivers. The HP iPAQ is no exception here, Linux still does not run on all models, especially not on the newest ones. Generally it has to be said that the newer the model, the more risky the use of Linux on it.

4.2.2 Running Linux on the iPAQ

Currently, two main distributions are available for the iPAQ:

- Familiar Linux focuses on making Linux installable on the 16 MB of memory the iPAQ has by default.
- Intimate Linux is based on Familiar Linux with the Debian packaging system "apt-get" and requires a memory extension of at least 140 MB [22].

Since Familiar Linux does not need memory extensions, it was the preferable distribution for this project. Nevertheless, although it fits onto only 16 MB it is a full Linux distribution currently based on a 2.4 kernel. Additionally, Familiar Linux provides its own packaging system similar to Debian's apt-get called "Itsy Package Management System" (ipkg), which eases the installation of new programs. The installation HOWTO, as can be found in [4] is very detailed and allows non sophisticated users to install Linux on the iPAQ.

Several problems occurred after the installation: At the time of the installation, the used iPAQ model, an h5450, was not yet fully supported by Familiar Linux. Neither the touchscreen, nor the buttons, nor the bluetooth chipset did work under version 0.7.1. With the release of 0.7.2-beta the touchscreen and the bluetooth chip were fully functional, the buttons still did not work properly. Additionally, the hardware sleep mode, essential for the mobile use, did not work on this model. The model h5550, used while writing this thesis does support the hardware sleep mode.

Familiar Linux uses the Gimp ToolKit (GTK) based GPE Palmtop Environment (GPE). So one option for the PersonalServer would be to use GTK as GUI library. As a second display manager the Qtopia based Open Palmtop Integrated Environment (OPIE) is supported, thus allowing Qt as GUI libraries as well. However, since the iPAQ's memory is very small it cannot be expected that both libraries are installed on the system. The decision was therefore to use the X libraries, since X is always installed when a display manager is used. GTK and Qt on the other hand depend on the system's configuration.

[2] http://www.gnu.org/licenses/gpl.html

All in all Linux runs well on the iPAQ, though many small problems make the daily use of it difficult. But the number of developers is big enough that it can be expected that those bugs and problems are fixed soon.

4.3 The C Implementation

The second implementation was done in the C programming language using the X graphics libraries for the GUI on the PersonalServer device. On the following pages reasons for C are given as well as an overview about the extra functionality the C implementation provides over the Java implementation.

4.3.1 Why C?

The C programming language is one of the most widespread languages, especially in the OpenSource domain, and even the Linux Kernel itself is written in C. This language has certain advantages compared to Java when it comes to speed and to the extensions of the language. For nearly all purposes there are already libraries, thus making it easier to use hardware or protocols.

C is usable on a great variety of platforms, especially embedded devices often only support C or pure assembler due to C's low hardware requirements. Of course, C is also available under PocketPC and under Linux on the iPAQ.

Problems occurring when programming in C are that, from a personal view, programming is more complicated and the overall handling is not as safe as Java. Java cares for memory allocations by itself, thus reducing the risk of writing into non-allocated memory blocks. Additionally, garbage collection is done automatically. C lacks of both technologies, making C programs a great security risk and a big source of errors for beginners.

4.3.2 The Plugin Framework in C

Loading applications at runtime is of equal necessity as in the Java implementation. The mechanism provided here is to compile an application as a dynamic library and then load it with the *dlopen()* function. As it is explained in [52], "*dlopen()* loads the dynamic library file [...] and returns an opaque "handle" for the dynamic library", therefore making it possible to access methods and variables in this library. So what needs to be done is to write an application, but instead of compiling it as self-running executable, it has to be compiled as a dynamic library. This is done with the *gcc* compiler switch *-fPIC* or *-fpic* as explained in [35]. *PIC*, Position Independent Code, allows the operating system to dynamically load a code section.

As in Java it is also necessary to provide a common interface for all applications to be able to access them from the PersonalServer. This interface is explained in Section 4.3.3. Additionally, each library has to provide a special method called *application_init*.

```
application_t* application_init ( void );
```

This method is not part of the application template (*application_t*) but is the first method accessed by the PersonalServer. It has to provide the instance of the application template to register it at the PersonalServer. All in all, the

plugin system is quite similar to the plugin system used in the mediaplayer XMMS, as it is explained in [30].

4.3.3 The Application Template

To allow the PersonalServer to access properties and methods of an application, the data type *application_t* was defined.

```
typedef struct
{
  char* name;
  int colormode;
  appgui_t gui;
  appmethods_t methods;
} application_t;
```

The *name* character string defines the name of the application. This variable is for example used by the task manager to display the list of running applications. The variable *colormode* is important to tell the PersonalServer which color mode is used in the application. As listed in Table 3.1 on page 17 less colors result in a shorter transfer time. However, the number of necessary colors has to be chosen by the application since it depends on the application's context. Currently, the watch supports RGB with a depth of 16 bits and 1 bit black and white color.

To allow the watch to use the GUI properly, the data type *appgui_t* was defined.

```
typedef struct
{
  int l_width;
  int l_height;
  int r_width;
  int r_height;

  Pixmap l_display;
  Pixmap r_display;

  GC l_canvas;
  GC r_canvas;
} appgui_t;
```

The first four variables listed here determine the size of the local display (on the iPAQ) and the size of the remote display (on the watch). Different to the Java implementation, where every application had the same fixed size, it is possible now to assign different display sizes to the application. Different display sizes allows for the possibility of showing more than one application at the same time: At the bottom of the LCD of the watch there is a clock, running parallel to the other applications which use the upper part of the watch. The same happens on the PersonalServer device where a panel is shown in the lower part of the screen.

The other variables can be used by applications which directly access the X library for display functions instead of the wrapper functions of the PersonalServer (explained in Section 4.3.4). They define both the canvas as well as the actual *Pixmap* where the pixels are stored in the X library.

From the view of the PersonalServer, the most important part are the values of the *appmethods_t* struct. This struct defines the functions the application implements.

```
typedef struct
{
  void (*init)      (void);
  void (*activated) (void);
  void (*cleanup)   (void);
  void (*press)     (int x, int y);
  void (*release)   (int x, int y);
  void (*key_press) (int charcode);
  void (*set_parameters) (char**, int);
  void (*refresh)   (void);
  void (*new_data)  (int socket, short mask);
} appmethods_t;
```

The functions listed here can be implemented by applications and are then invoked by the PersonalServer. The *init()* function is used for initializing the application's variables, whereas *cleanup()* can be used for deallocating them just before the application shuts down. Different to the Java implementation it is differed between *press()* and *release()* on mouse clicks. Additionally, keystrokes can be evaluated by using *key_press()*.

As mentioned before, applications have to be compiled as a dynamic library. The downside of this is that applications cannot have a *main()* function providing them with command line parameters. As a substitute the function *set_parameters()*, which's parameter definition is equal to the main function, can be used to set certain prerequisites. However, this function is not called automatically but could be used to copy data from one application to another.

A special attention has to be turned to *refresh()* and *new_data()*. Since the PersonalServer is single threaded applications have to be pushed regularly to allow repetitious tasks. This is done with the function *refresh()*. A good deal of applications will need a network connection, all of those have to be nonblocking, otherwise the PersonalServer would stall. Fetching data from nonblocking sockets is done with the *select()* function. It is hard to synchronize more than one *select()* call within one program, so this task is done by the PersonalServer. It checks the open sockets for available data and then calls the *new_data()* function of the corresponding application to notify it. The application then can read data from the open socket without blocking the program.

4.3.4 The C PersonalServer

The architecture of the PersonalServer consists of three main elements (see Figure 4.4): The main PersonalServer (*xps*), *libxps.so* and the dynamic libraries of the applications, in the figure listed as *libMyApplication.so*. The PersonalServer

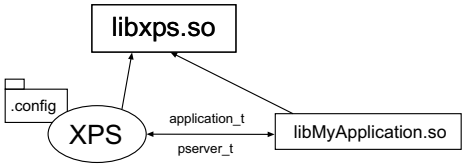


Figure 4.4: Simplified diagram of the PersonalServer architecture.

and *libMyApplication.so* are accessing the library *libxps.so* for redrawing functions. The additional library is necessary to be able to complete the link process after compiling the program.

To be able to access properties and methods of each other, both the PersonalServer as well as the application have to exchange structs. The struct representing the application is called *application_t* and was described in the previous chapter, the struct representing the PersonalServer is called *pserver_t*.

The PersonalServer's basic settings can be configured with a simple configuration file, such as the size of the remote display or the network configuration. Applications can use the same configuration file, methods are provided to easily access the entries. The syntax of the file is quite similar to the style many programs use under Linux, consisting of a key-value pair for each entry. Comments can be made by using the # character.

```
# Remote Watch specs
Host 10.0.0.1
Port 14234
Height 160
Width 120
Packetlength 150

# PersonalServer specs
Appdir /home/whot/code/HPWatch/XPS2/Apps/
Internal /home/whot/code/HPWatch/XPS2/internal/
[...]
```

This example shows how to define the dimensions of the displays and the network settings for the watch. The example also shows two important values: The path to the application directory (*Appdir*) and the path to the directory for internal applications (*Internal*). The PersonalServer uses the *Appdir* to search for and to load applications, an application can therefore be installed just by copying the library into the *Appdir* folder. The difference to the *Internal* folder is only a slight one: this folder contains special applications which are started in a different style, such as the clock and the panel. Generally, the concept is better modularized than the Java implementation. All applications, including task manager, application starter and even the clock and the panel are based on the application template described in Section 4.3.3 and can be exchanged easily.

The startup of the program is quite similar to the Java concept, the watch display is initialized, then the GUI on the PersonalServer device. It differs though when it comes to the main loop. In Java this is done within the AWT library and each application runs in its own thread. Here, this task is done by

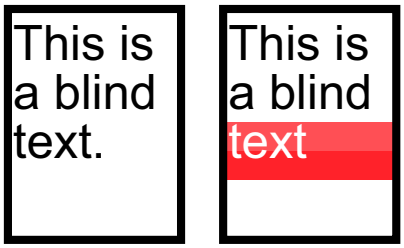


Figure 4.5: Sample LCD update. If the word "text" changes color, only a part of the screen is redrawn. In this example this part consists of two boxes.

main_loop() in the PersonalServer program. The reason for this single-threaded approach was to allow the use of different, less powerful devices in the future. While a device supporting Java usually supports threading too (except devices implementing the Java 2 Micro Edition), this is not necessarily the case with C. In the future, the iPAQ could be exchanged against a specific device which might not be supporting threads. Since the code accessing the X library is separated from the rest a different graphics library can be used in the future too.

What happens now in *main_loop()* is that the PersonalServer first checks for events from the X server, then it checks the open sockets for pending data. Finally, it calls *refresh()* on all applications to allow them to update their status.

A difference between the two implementations is how a screen redraw is done. The C implementation optimizes the redraw by just sending those parts of the screen which have changed instead of the whole screen. This saves transfer and processing time on the watch, resulting in a faster screen update. However, it is not possible to just transfer the changed pixels since this would cause a large overhead by the network protocols. It can be assumed though that if one pixel changes state it is very likely that pixels close to this one change their state too. What is done is to transfer blocks of twenty pixels height and the width of the LCD, hoping that changed data will mainly occur in lines. As shown in Figure 4.5 the first block starts at the first changed pixel. This results in a better relation between packet data and protocol overhead.

Some effort has been put into making programming easier for application programmers: Several graphic functions such as assigning colors to the display, drawing lines and similar are wrapped by the PersonalServer:

```
void create_canvas (application_t *app);
void free_canvas (application_t *app);
void draw_line_local (application_t *app, point_t start, point_t end);
void draw_line_remote (application_t *app, point_t start, point_t end);

void clear_local (application_t *app);
void clear_remote (application_t *app);

void draw_text_local (application_t *app, char* text, point_t where);
void draw_text_remote (application_t *app, char* text, point_t where);
```

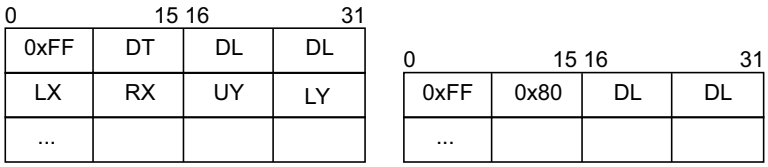


Figure 4.6: ITP start packet (left) and continuation packet (right).

```
/* binary data 1 bit per pixel */
#define ITP_COLOR_BIN 11
/* binary format, run length encoded */
#define ITP_COLOR_BINRLE 3
/* rgb 16 bit */
#define ITP_COLOR_RGB16 10
/* rgb 16 bit with RLE */
#define ITP_COLOR_RGB16_RLE 9
```

Figure 4.7: Supported data types on the watch (extract from *ITPConstants.h*).

```
void draw_rect_local (application_t* app, point_t where, point_t dim);
void draw_rect_remote (application_t* app, point_t where, point_t dim);

void set_local_color (application_t *app, color_t color, int mode);
void set_remote_color (application_t *app, color_t color, int mode);
```

These functions represent the most often used methods to draw on the canvas. Apart from reducing the code by several lines these wrapper functions make the application independent from the graphic library used on the PersonalServer. However, applications can still use direct access to the X library if they want to or need to.

4.4 The Image Transfer Protocol

To transfer images to the watch, a protocol is needed specifying the content of the transmitted packets. The Image Transfer Protocol (ITP) was developed to suit this situation.

The main goal of the ITP is to divide an image data stream into a sequence of packets which can be transmitted over the underlying network and then reassembled to display the data with the correct windowing information. The current version ITP v3 defines two different packets: a start packet and a continuation packet. The main difference between those two packets is the windowing information as shown in Figure 4.6.

In a start packet the header consists of eight bytes. The first byte defines a control sequence to recognize a ITP packet, the second byte defines the data type of the image in a range from 0 to 127. Currently the data types listed in Figure 4.7 are supported by the watch.

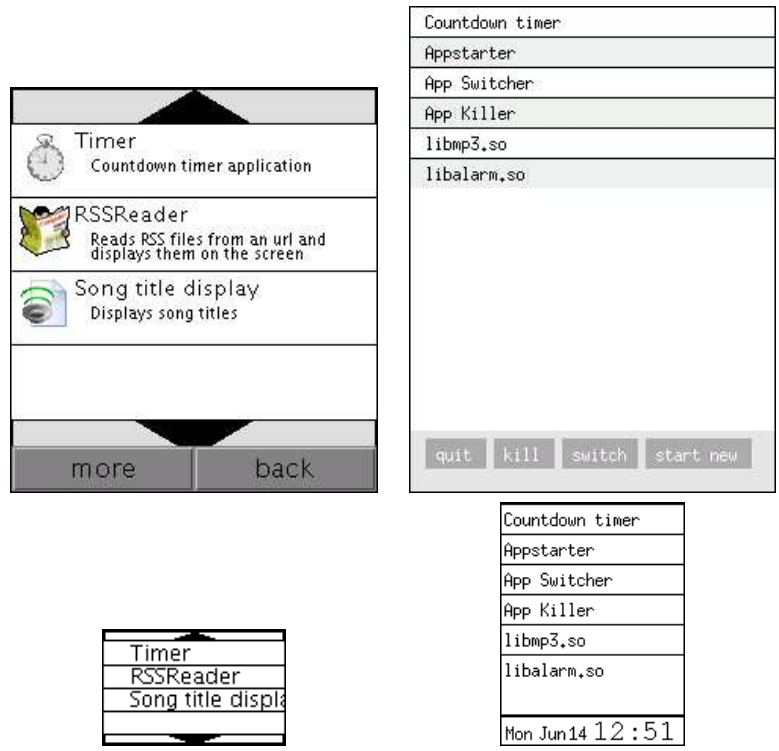


Figure 4.8: The task manager showing available applications in the Java (left) and the C Implementation (right). The images below represent the matching counterparts on the watch.

Bytes three and four of the ITP packet define the total length of the payload as a 16 bit value, whereas byte three defines the most significant bits (MSB). Since the watch provides methods for windowing, thus specifying a certain area on the display to fill up with data, each ITP start packet has to contain these window information. This is done in the bytes five to eight: LX and RX define the left and the right border, UY and LY define the upper and the lower border.

A continuation packet can skip this window information, reducing the header to only four bytes. Additionally, the data type does not have to be provided, since it is assumed that it is the same as in the latest start packet. Instead, the MSB of byte two, the so-called "Continuation Flag", is set to signal that this packet is a continuation packet. Byte three and four are, just as in the start packet, used for specifying the payload length.

4.5 Example Applications

In this section some applications are described that were implemented in either the C or the Java framework (or both). Each of these applications is an example for a technique how to process certain input data. In Section 4.5.1 a timer application is described which connects to household devices. An appli-

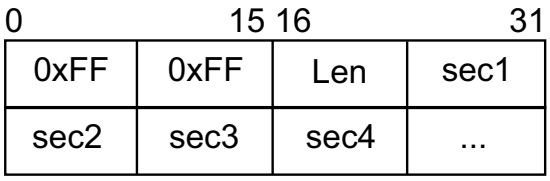


Figure 4.9: Structure of a timer packet. Bytes one and two are a sequence of $0xFF$ to identify a timer packet. Byte three contains the length of the identifier message (max. 256). Bytes four to seven contain the number of seconds with byte four as most significant value. Bytes eight to the end of the packet contain the identifier string.

cation showing how to connect the PersonalServer with an external application is demonstrated in Section 4.5.2. Section 4.5.3 explains how applications can fetch live data from the Internet. In contrast, the application discussed in Section 4.5.4 demonstrates file based input data. Finally, Section 4.5.5 shows the use of high data video streams. Figure 4.8 shows a screenshot of the taskmanager listing installed applications.

4.5.1 Direct Hardware Communication - The Timer

As shown in Section 3.1 on page 10, the PersonalServer should be able to communicate directly with other hardware. The decision was to implement a timer, useful for many household devices. This shows that future household devices only need slight modifications to be able to make use of the PersonalServer.

The use of a countdown timer is ambiguous as for example a coffee machine or a stove could use it to show the remaining time of food to be finished. To do this, the device needs to send a certain message to the timer application, which then shows the remaining time. The users can meanwhile focus their work on something else, the countdown is displayed on their watches.

An extension to existing household devices is fairly simple. Cheap microcontrollers are able to send these simple messages over a bluetooth network. The Bluetooth chips themselves are fairly inexpensive. More and more household devices already have network connectivity and can be expected that in the future most devices are connected. For example, since 2000 it is possible to buy internet connected washing machines [55], using those it is easy to send messages to a PersonalServer if the server has a public IP address.

The message is kept very simple. In the implementation a message has seven or more bytes, where the first two bytes are used as a identifying sequence. Four bytes are used to define the time in seconds, thus allowing a maximum of over four billion seconds or over 1.2 million hours. This enables other devices with the need for a longer countdown to use the same message type. In addition to those bytes the packet specification allows an identification string of a maximum length of 255 characters. The full packet format is shown in Figure 4.9.

The timer application is not necessarily locked to the hardware. Since the message type is abstract software can use the same application to display data. An example for this would be a download, where the remaining time is displayed, while the user is not sitting in front of the computer.

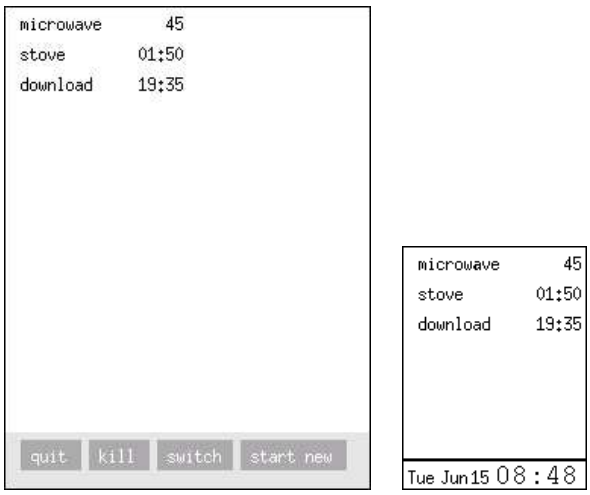


Figure 4.10: The timer application showing three different timers

The timer application is built to show more countdowns at the same time, sorted by finishing time (see Figure 4.10). However, the application is kept very simple and contains only a few hundred lines of code. The plugin API is a wrap around some of the most used X functions so that both GUI and network initialization are only a few lines of code.

4.5.2 Communicating to other Applications - MP3 Title Display

Figure 3.4 on page 14 shows that other applications should be able to communicate with PersonalServer applications. One example to show this is the MP3 Title Display which consists of both a PersonalServer application as well as a plugin for XMMS. XMMS is a popular MP3 Player for Linux, which provides a rich API for extensions. The plugin retrieves information about the current song and passes it on the PersonalServer application, which then displays the information as can bee seen in Figure 4.11.

Applications communicating with other, non-PersonalServer applications, can be used in a large application domain. A similar system as the MP3 Title Display could be used to connect the default mail client with an application on the PersonalServer. It would be possible to get information about the users' mailbox without the need of implementing protocols like POP3 or IMAP. Modern e-mail clients as Mozilla provide a plugin system which would allow that. The communication between the PersonalServer applications and the external application could happen either via networking or via a common shared library. For simplicity and easier portability the MP3 Title Display communicates with the XMMS plugin via network. This allows the MP3 Title Display to communicate with other plugins such as a port to Winamp as well.

The communication between MP3 Title Display and the XMMS plugin is held simple. Each second the plugin fetches information about the current song, information such as the artist, the title, length of the song and the position in

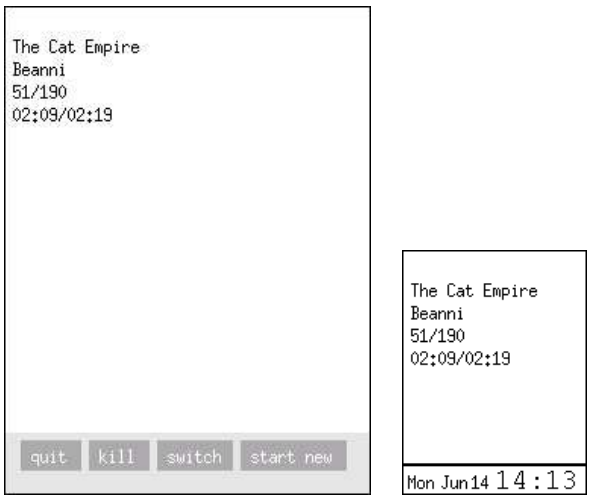


Figure 4.11: The PersonalServer (left) and the watch (right) displaying basic song information.

the playlist. Each of those is sent as a single UDP packet, thus allowing the plugin to be active even though no PersonalServer application is connected. This is an advantage if the user switches PersonalServers often.

Within the plugin information is gathered every second and converted to strings. A UDP connection is used to broadcast this information. Each string of information is sent as a single packet. Therefore, a full information collection consists of seven packets, as shown in the following example:

```
Packet 1: ARTIST: Muse
Packet 2: TITLE: Time Is Running Out
Packet 3: PLAYLIST: 187
Packet 4: POSITION: 2
Packet 5: TOTALTIME: 236
Packet 6: TIME: 34
Packet 7: PLAYING: 1
```

This example shows that the current song is “Time Is Running Out” from the band “Muse” at position two of 187 totally in the playlist. 34 seconds have elapsed already, of 3:56 min total. The last line defines whether XMMS is playing (1) or stopped/paused (0).

4.5.3 Live Data Streams - RSS Feeds

Live data streams are vitally important to many businesses today. These can consist of either the latest stock quotes or just the latest headlines of the IT industry news.

A widely used method is RSS [20] which is a substandard of XML and supported by a large number of news sites. Even stock courses, often proprietary protocols, are available via RSS on some websites.

Figure 4.12: RSS reader in the Java implementation with an additional dialog box.

A big advantage in RSS' XML structure is that it is possible to use pack-and-go XML parsers without the need to implement a parser by hand. This also eases the use of more streams at the same time. Especially free RSS feeds often have a low update rate. However, it is possible to combine more than one feed with different update times to get a better overall update frequency.

The output of an RSS news feed can be either graphical or textual, depending on the user's context and the information itself. If it is necessary to keep an eye on stock prices over a longer time interval a graphical interface such as line diagrams are a good solution. However, if there are too many lines on the diagram it becomes fuzzy and difficult to read. On a display as small as the one on the watch it is hard to provide a proper legend to the diagram. In this case it is better to display the information in text format.

For a pure RSS news feed the textual representation is mandatory. The decision is then to either represent the data as a ticker or as fixed text blending from time to time. With fixed text the problem on small displays is that longer headlines are difficult to render. It is possible to display about 15 to 20 words on the display, depending on the length of the words. Text has to be optimized for the small display size but general newsfeeds are not. Especially long words would have to be separated in two but to follow grammatical rules a complex algorithm is necessary here. This makes the fixed text less attractive.

A ticker allows it to display more feeds than one at the same time, but an animation on the watch, which is visible most of the time, can be distracting. However, newsfeeds do require the attention of the user and are not a background program, so this problem can mostly be ignored. Another problem with the ticker is that it is hard to find a relation between too fast to read and too slow to wait for the next newsline. The best solution to this is to let the user define the scrolling speed as every user will find a different speed optimal. As the application needs to remember that setting over sessions it needs to be defined in the configuration file. Figure 4.12 shows an RSS newsticker with scrolling text.

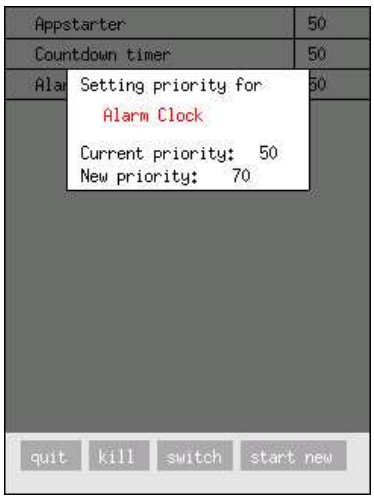


Figure 4.13: Defining a priority for an application.

4.5.4 Personal Information Management - Alarm Clock

In contrast to the applications mentioned above the Alarm Clock shows that an application does not need a permanent network connection. Instead, the Alarm Clock uses a data file to retrieve the user's schedule. This data file is a exported file from the popular PIM application KOrganizer. The Alarm Clock uses the priority redraw mentioned in Section 3.3.3 to display upcoming events (see Figure 4.13). In addition to that, it uses the PersonalServer's APIs to fetch information from the global configuration file.

```
char *filename = get_config ("Alarmfile");
read_alarms (filename);
```

Using those two lines, the PersonalServer returns the configuration tag with the key "Alarmfile" from the configuration file. By allowing applications to use this file it is easier to configure and maintain a server. Synchronizing the Alarm Clock with the user's organizer is simple. KOrganizer exports .ics files, which can then be copied onto the PersonalServer. Moreover, if the PersonalServer device is the same device KOrganizer is used on, the Alarm Clock can directly use the .ics file KOrganizer uses internally without forcing the user to synchronize manually. Applications using the same method to retrieve data could additionally use a file change notification mechanism to reload changes.

If the Alarm Clock is active, it shows the next five entries within the next two hours. Thirty minutes before one scheduled event is due, it will be displayed with high priority to notify the watch user.

This application shows that application are possible without a permanent connection to external devices and/or networks. This is necessary for applications used in mobile situations, for example in public transports where no internet connection is available.

4.5.5 Video Streaming - Baby Monitor

With the upcoming third generation of mobile phones video telephony was thought as the compelling application. One way of displaying the video stream would be to send it to the LCD of the watch, so that users can use their handset as audio input device and the watch as video output device. Video streaming is useful in other domains as well, such as doorbell cameras or baby monitors to survey a child while it is sleeping in its bed.

However, there are some problems with video streaming on the watch. One is that in the case of a bidirectional video stream, such as video telephony using the watch disables one stream. There is no camera mounted on the watch, so the peer would not receive a video stream. This might not be important in most situations, though it is in business conversations, where this would be considered impolite.

Another problem occurring especially with video data is the high data density, which requires greater bandwidths then a technology like Bluetooth can support. Only with complex codecs and low framerates it is possible to transfer video without time delay. The watch uses very simple hardware, which makes it impossible to decode codecs such as MPEG-4 or DivX. Therefore, the data flow between watch and PersonalServer cannot be compressed with codecs. Additionally, in the case of video telephony the video would have to be transferred via Bluetooth to the PersonalServer and then again via Bluetooth to the watch since most mobile phones only support Bluetooth but not WLAN.

So video streaming is more useable if the necessary framerates are very low, as it is the case with baby monitors. Though the camera might provide 25 fps or even more, it is possible to drop frames down to a framerate of 1 fps or even less without losing much of the information. For observing a sleeping baby, one frame per second is enough information.

To further decrease the data rate it is necessary to reduce the amount of colors and compress the data. Run length encoding [33] is a simple compression method possible on simple hardware and is more efficient if the number of colors is reduced to 256 or less.

Dropping frames to reduce the framerate has to be done as early as possible. If one frame needs one second transfer time, the last frame of the first second of a 25 fps stream would arrive after 25 seconds have passed. Event worse, this delay is cumulative. So software is necessary at the first instance to check the transfer rate and drop frames dynamically. In the implemented version the camera is controlled with the Java Media Framework. The captured frames are cut down to the size of the LCD on the watch and then sent to the PersonalServer. To avoid the cumulative delay, a simple double buffer is used for sending the frames. While the first frame is transferred, each following frame is saved into a buffer, overwriting the one before. When the transfer is finished, the current frame in the buffer is copied into the send buffer and transferred. To reduce processing time on the PersonalServer the frames are already packed in the format the watch is able to receive. Therefore, the PersonalServer's task is only to route the packets, using the same double buffering technique as mentioned above.

Chapter 5

The Prototype

This chapter gives an overview about the prototype which was used during the development. This prototype, as shown in Figure 5.1, was manufactured by the Hewlett Packard Research Labs (HPLabs) in California, US. It was tested with the C implementation of the PersonalServer framework as described in Section 4.3. Most of the code running on the watch was developed by John Ankcorn.

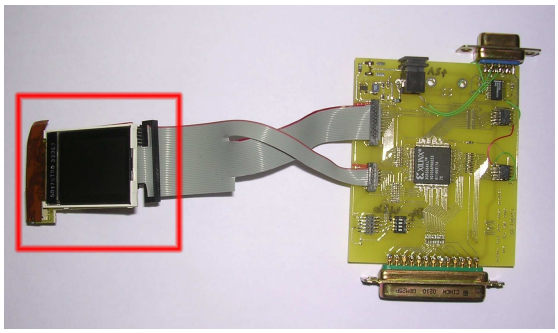


Figure 5.1: The prototype developed from the HPLabs in California (US). The red square marks the actual prototype, the board on the right is used for flashing and debugging.

5.1 The Hardware

An overview about the hardware of the watch's prototype is now given. Mainly, the watch consists of three elements: the display, a Bluetooth chipset and a programmable microcontroller.

5.1.1 The Display

The display used on the prototype is an Epson L2F50176T00 (see Figure 5.2). The display's resolution is 120x160 pixels at a screen size of 2x2.6 cm (1.3 in diagonal). This results in roughly 152 DPI. The LCD supports RGB with a

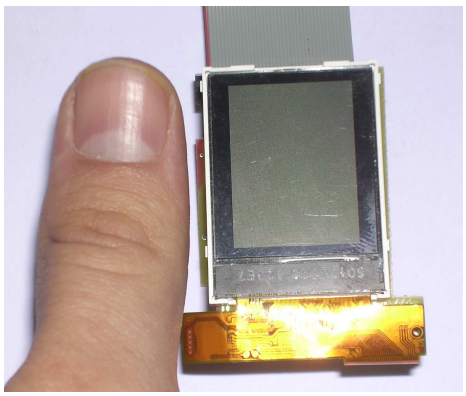


Figure 5.2: The watch's display.

color depth of 24 bits per pixel, though in the current implementation only 1 bit black and white and 16 bit RGB are supported.

The display is hooked up via two connections, one used as a data line, the other one as control line. Writing of image data has to be done serialized, as the following code segment shows:

```
/* write a 16 bit RGB value to the display */
void lcd_data_write_pixel (unsigned int data)
{
  LCD_DATA = (data >> 8) & ~7;
  LCD_CONTROL += LCD_X1; /* toggle X1 flag to cause clock to LCD */
  LCD_DATA = (data >> 3) & ~3;
  LCD_CONTROL += LCD_X1; /* toggle X1 flag to cause clock to LCD */
  LCD_DATA = data << 3;
  LCD_CONTROL += LCD_X1; /* toggle X1 flag to cause clock to LCD */
}
```

Between two data bytes the *LCD_CONTROL*, the control line, has to be written with the value *LCD_X1* to let the LCD internally switch to the next data byte. The maximum refresh rate is around four frames per second.

5.1.2 The Bluetooth Chip

The used Bluetooth chipset is a Mitsumi WML-C09NBR without antenna. Like the LCD, the Bluetooth chip is connected over a Universal Asynchronous Reciever-Transmitter (UART) interface, the maximum data rate over Bluetooth is 721 kbps [45]. The WML-C09NBR is a class 2 chip, allowing a range of up to 10 m. Tests showed that in combination with a standard USB Bluetooth adapter the maximum distance is just three to four meters.

5.1.3 The Microcontroller

The microcontroller used on the watch prototype is a Texas Instruments MSP430 and quite common in low power devices. The specific model name is F1491,

which has 64 KB ROM for program code and 2 KB RAM for dynamic variables [59]. Additionally, it has two serial UARTs for connecting to the Bluetooth chip and to the LCD. The 16 bit processor is running with 4 MHz, the whole controller only needs 280 μA in active mode, making it optimally fitted for this project.

The MSP430 is programmable with C code and a port of gcc for the MSP430 series is available. However, programming on the MSP430 is different to desktop programming since even common methods such as *print()* are not available. Additionally, no memory management is available, arrays can therefore not be dynamically allocated at runtime.

To bring the code onto the watch, additional hardware is necessary (see Figure 5.1). In the future, the software will be preinstalled on the watch and the watch will be sold without this additional device.

5.2 Code on the Watch

To provide the requested functionality on the watch it is necessary to run software on the microcontroller. This software consists of two parts: One part is responsible for initializing the microcontroller and the LCD as well as implementing the network stacks for the Bluetooth connection. This part will be discussed in Section 5.2.1. The other part is the project related part which decodes ITP data packets and displays them on the screen. This is explained in 5.2.2.

5.2.1 Setting up the Watch

To make the watch usable it is necessary to initialize the MSP430, the LCD and the Bluetooth chip. The code discussed in this Section was implemented by John Ankcorn from the HP Labs in California and not by the author of this thesis.

The plan was to communicate via UDP or TCP between PersonalServer and watch, which results in the need for network stacks and of course the stacks for Bluetooth communications. All in all, eight network stacks had to be implemented to be able to establish a Bluetooth connection and send data to the watch. These stacks allow the PersonalServer to have a direct connection to the watch by using the Linux *Personal Area Network Daemon (pand)*.

Additionally to the network stacks, John Ankcorn provided methods to access the display in a convenient way:

```
void lcd_window (int, int, int, int);
void lcd_data (int len, int rgb);
void lcd_data_init ();
void lcd_data_flush();
void lcd_data_write_pixel (unsigned int);
```

The *lcd_window()* function allows the definition of a certain area on the screen which can then be used to draw on. Therefore, it is possible to only send the parts which have to be updated instead of the whole display, reducing the network transfer time and increasing the overall execution speed. The *lcd_data()* function writes an RGB value to the display with a repetition count as provided

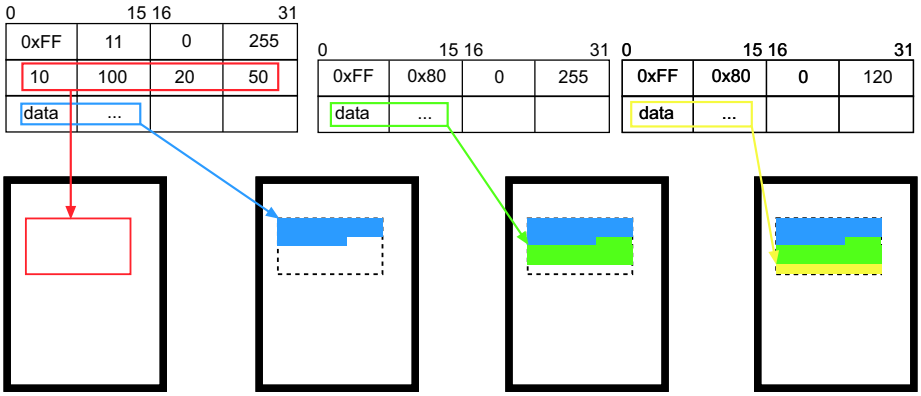


Figure 5.3: Example for the ITP packet processing order. The windowing information from the ITP start packet is used to call *lcd_window()*. Then the data is taken and sent to the display. The data of the following packets is displayed subsequently to the last pixel of the previous packet.

in the second parameter, whereas *lcd_data_write_pixel()* writes only one value to the screen. *lcd_data_init()* and *lcd_data_flush()* are control functions used for flushing the display and therefore displaying the data.

This code, the network stacks and some initializing code for the MSP430, provide a setup for the watch which can be used by every application which wants to display data on the screen.

5.2.2 Displaying Data

On reception of a valid data packet the method *lcd_app_data()* is invoked. Within this method the network headers (IP, TCP, UDP) are filtered and *draw()* is called with the remaining data. This method is the core of the watch's code, it decodes ITP data and displays them on the LCD.

The approach is simple: First, there is the check whether the data is an ITP data or not. This is done by checking the ITP starting sequence. If the packet is an ITP start packet, the windowing function *lcd_window()* is called. Afterwards, the remaining data is written to the display as shown in Figure 5.3.

However, during the first tests problems were encountered with the long transmission times. To avoid this problem Run Length Encoding (RLE) was introduced. This compression technique works well, especially on images with big areas using the same color, quite common with a normal GUI. Additionally, RLE has very little memory overhead, thus making it useable on even small microcontrollers such as the MSP430. RLE uses only two additional variables and does not need to save results while decoding.

The actual code on the watch for RLE is mostly the same as with unencoded packets. Instead of displaying value once, an RLE encoded value is displayed according to its repetition count.

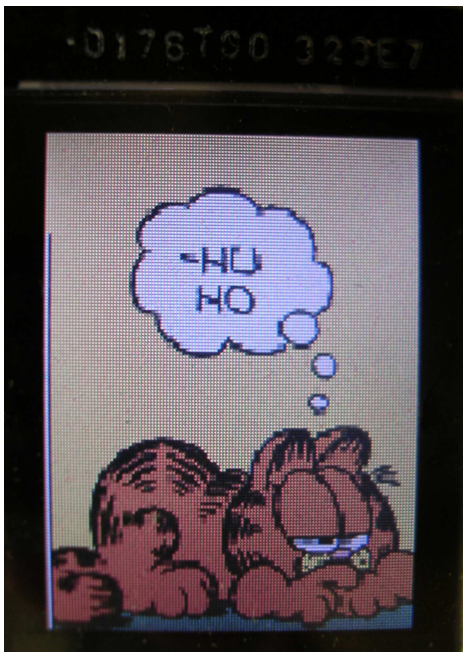


Figure 5.4: A comic strip shown on the display. The LCD's origin is in the upper right corner which caused the images to be mirrored on the y-axis in the first versions.

5.3 Issues with the Watch

Two issues occurred while using the watch in the current configuration, both regarding the microcontroller. Since the prototype is using a power supply unit instead of a battery pack no expectations regarding battery life can be made yet.

5.3.1 Processor speed

The bigger issue is the processor speed. Nearly every program logic already runs on the PersonalServer but the MSP430 is still a little bit too slow. The main problem here is that the incoming data has to be copied into buffers several times. If there is too much data coming over the Bluetooth interface, the MSP430 is not able to process the data fast enough. This causes existing buffers to be overwritten, resulting in a data inconsistency and often in a processor reset. A faster processor is necessary here.

5.3.2 Processor memory

Another problem is the lack of memory on the MSP430. The network stacks already need a large amount of memory and all data has to be stored in more than one buffer while processing the network stacks. To fit into the memory then, all buffers have to be small. This again causes the packets sent over the network to be small enough to fit into the buffers, resulting in a big network

overhead which sometimes exceeds 50% of the packet. Although extending the memory would allow bigger buffers and therefore larger packets, the processor speed has to be enhanced too, otherwise it would not be able to process the packets fast enough.

More memory would allow other options such as font support. Instead of a huge amount of pixels, only characters and the coordinates have to be transferred. This reduces the amount of data to only a small portion.

Chapter 6

Review and Future Works

This chapter now first gives a critical review on the concepts and the actual implementation and then a view in the future what might come next.

6.1 Critical review

It is obvious that using a multi-purpose data display instead of a watch has advantages. But there are drawbacks too: In today's stressed out world displaying data on the users' wrists can put even more pressure on them. While an iPAQ can easily be ignored, this is hardly possible with a display on the wrist. So this always visible data could increase the stress for the users. A solution to this can only be to carefully select the applications running on the watch, so that those running are a relief to the users instead of a new source of stress.

A hard thing is to find a compelling application for the watch. Most applications suitable for the watch already exist on the PDAs, so the advantage of the watch is just to have an additional display. This is even more so since the watch does not have any interaction possibilities, a major flaw in the current setup. An additional interaction device, either directly on the watch or as separate device, would increase the usability of the watch.

The size of the display is slightly too small to display a reasonable amount of text. It is big enough to display small and informational graphics and even comic strips can be shown but when it comes to text the resolution is not enough. This is partly because of the portrait format and partly due to the small resolution of only 120x160 pixels. A display with a higher resolution is needed here.

One problem occurring due the concept of this project is that the watch itself does not have sufficient functionality. A data display can be of use in many situations, but if this data display is rendered useless when the connection to the PersonalServer is lost, the whole approach has to be questioned. Additional hardware on the watch is necessary to allow at least timekeeping functions if there is no connection available.

Other problems occurred due to the little experience of the author of this thesis. Programming in C was a practically new field, especially under Linux. This explains many problems which occurred during the development and also the rather large time budget needed for this project. For example, several weeks have been spent on rewriting the code on the watch and adjusting it to fit the

PersonalServer's needs better. A venture which finally failed.

The decision to use Linux is hard to judge. While fans of this operating system will welcome this approach, critics might point out that this is a no-future option since Linux is not the standard on PDAs and currently it does not look as if that would change in the next years. The future of Linux on the iPAQ remains unclear too, since it depends on a number of volunteers whose work is made difficult due to the lack of hardware vendor support. However, in the opinion of the author the knowledge produced with this project justifies the use of Linux.

6.2 Future Works

It can be expected that in near future more projects will focus on producing a multi-purpose watch. Microsoft will expand the service for the Wrist Net watch to Europe, as testing has already begun [15]. The miniaturization of the PDAs promises an interesting future in this field too.

If the approach used in this project will spread is difficult to foresee. However, it is to hope that future projects will provide an interface to use the watch as data display from an external device. To make this project fully usable, several changes have to be made: The microcontroller should be replaced by a different, more powerful model to allow faster processing and the watch should be equipped with additional hardware for timekeeping as was pointed out in the section before. Additionally, user tests have to be made when the prototype is finished to find further drawbacks. Finally, a re-implementation should be considered to match the new insights.

Appendix A

CD-ROM Content

File System: ISO 9660

Mode: Single-Session (CD-ROM)

A.1 Thesis

Pfad: /

thesis.dvi	Thesis (DVI file)
thesis.pdf	Thesis (PDF file)
thesis.ps	Thesis (PostScript file)

A.2 Online resources

Path: /bibliography

This folder contains all web resources used for citations in the thesis. All subdirectories represent a group of resources with similar topics. A detailed description of the folder structure can be found in /bibliography/README.

A.3 Implementation

All folders contain a README file which describes the content and how to compile, install and use the specific program.

Pfad: /implementation

LICENSE	License for the PersonalServer implementations, the application running on the OnHand PC and the display emulator for the watch.
GPL	GNU General Public License definition. This license applies to the XMMS plugin.

OnHand/	This directory contains the source code for using the Matsucom OnHand PC as display emulator instead of the watch and a Java UDP to SerialPortPort wrapper.
OnHandEmulator/	This directory contains the source code for a Java program emulating the Matsucom OnHand PC. It can be used in connection with the Java implementation of the PersonalServer.
PersonalServerC/	This directory contains the source code of the C implementation of the PersonalServer. The source was tested on an x86 machine (using gcc) and on a StrongARM based iPAQ (using arm-linux-gcc). This directory includes the man page for the PersonalServer and seven example applications.
PersonalServerJava/ . . .	This directory contains the source code of the Java implementation of the PersonalServer. This implementation is using the Matsucom OnHand PC as display emulator as remote display. This directory includes six example applications.
WatchDisplayEmulator/ .	This directory contains the source code for the display emulator which can be used instead of the watch.
XMMSPluginNetdisplay/	This directory contains the source code for a XMMS plugin to send song information over UDP packets to a remote host. This program is licensed under the terms of the GNU General Public License.
watch/	This directory contains the source code for the software running on the watch prototype to receive and display ITP data packets. This software is mostly copyrighted by the Hewlett Packard Labs in California.

A.4 Additional software

Pfad: /software

arm-libs.tar.bz2	This file contains the libraries copied from the Linux installation on the iPAQ. Due to linking problems with the libraries delivered with the toolchain these libraries here should be preferred when compiling applications for Familiar Linux 0.7.2-beta.
arm-linux-toolchain.tar.gz	This file contains the complete toolchain (version 3.2.3) for compiling code for the

	StrongARM processor used on HP iPAQs.
inq_respond.tar.bz2 . . .	This file contains the vanilla program provided by John Ankcorn. This program can be used to let the watch respond to Bluetooth inquiries (hcitool inq) and reveal the hardware address of the device.
watch_toolchain.tgz . . .	This file contains the complete toolchain for compiling code for the MSP430 used on the watch and John Ankcorns example codes.

Bibliography

[1] About the Java Technology. URL, http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.htm%l, April 2004. Copy on CD.

[2] Bluetooth range in relation to different power classes. URL, http://www.palowireless.com/infotooth/knowbase/general/10.asp, March 2004. Copy on CD.

[3] Data Link - FAQ. URL, http://www.timex.com/datalink/faq.html, May 2004. Copy on CD.

[4] Familiar v0.7.2 installation. URL, http://familiar.handhelds.org/familiar/releases/v0.7.2/install/, April 2004. Copy on CD.

[5] FAQs - iPAQ Pocket PC h5100 and h5500 series. URL, http://h10010.www1.hp.com/wpa/vat/genPage.do?vacpage=pocketpc/faq_h5550%.vad&wf=WF05a&segment=sm&country=us&lang=en&fpoid=215348-64929-215381-314903-f%44-322916#battery, March 2004. Copy on CD.

[6] Fossil Wrist PDA Watch cancelled. URL, http://www.i4u.com/article1059.html, May 2004. Copy on CD.

[7] Gartner says Palm OS licensees accounted for 51 percent of PDA shipments and 41 percent of worldwide revenue in second quarter of 2003. URL, http://www3.gartner.com/5_about/press_releases/pr15aug2003b.jsp, March 2004. Copy on CD.

[8] Glossary of terms. URL, http://java.sun.com/docs/books/tutorial/information/glossary.html#inter%face, April 2004. Copy on CD.

[9] Internet zum Burger. URL, http://www.heise.de/newsticker/meldung/46446, April 2004. Copy on CD.

[10] Ironman Data Link USB. URL, http://www.timex.com/bin/detail.tmx?item=753048051656, May 2004. Copy on CD.

[11] JAR files. URL, http://java.sun.com/products/javabeans/jar.html, April 2004. Copy on CD.

[12] Klimt - the Open Source 3D graphics library for mobile devices. URL, http://studierstube.org/klimt/index.php, June 2004. Copy on CD.

[13] Linux compatibility. URL, http://www.linux.org/docs/beginner/platforms.html, April 2004. Copy on CD.

[14] Microsoft launches Smart Personal Object Technology Initiative. URL, http://www.microsoft.com/presspass/features/2002/nov02/11-17SPOT.asp, May 2004. Copy on CD.

[15] Microsoft SPOT watches to go on sale in January. URL, http://www.infoworld.com/article/03/12/16/HNmsspot_1.html, May 2004. Copy on CD.

[16] Microsoft's Gates touts concept of seamless computing during pre-CES keynote presentation. URL, http://www.cesweb.org/press/news/release_detail.asp?id=10388, January 2004. Copy on CD.

[17] onHandPC. URL, http://www.shoplite.com/onhand.htm, March 2004. Copy on CD.

[18] PalmOS vs. PocketPC. URL, http://handheld.medicine.dal.ca/introduction/, March 2004. Copy on CD.

[19] PocketPC PDAs to surpass PalmOS PDAs in 2005? URL, http://www.linuxdevices.com/news/NS6990962584.html, March 2004. Copy on CD.

[20] RSS 2.0 specification. URL, http://blogs.law.harvard.edu/tech/rss, April 2004. Copy on CD.

[21] Sun ray ultra-thin clients. URL, http://wwws.sun.com/sunray/index.html, June 2004. Copy on CD.

[22] The intimate project. URL, http://intimate.handhelds.org/index.html, April 2004. Copy on CD.

[23] Timex 50-Lap Ironman Triathlon with Data Link System. URL, http://products.consumerguide.com/cp/electronics/review/index.cfm/id/19%838, May 2004. Copy on CD.

[24] Trail: The Reflection API. URL, http://java.sun.com/docs/books/tutorial/reflect/, April 2004. Copy on CD.

[25] What is an Interface? URL, http://java.sun.com/docs/books/tutorial/java/concepts/interface.html, April 2004. Copy on CD.

[26] WLAN-Karte mit 100 mW Sendeleistung von Allnet. URL, http://www.golem.de/0304/25188.html, March 2004. Copy on CD.

[27] Wrist Net Reference Guide. URL, http://www.fossil.com/text/content/tech/downloads/wristnetreferencequid%e.pdf, January 2004. Copy on CD.

[28] Wrist Net Round FX3001. URL, http://www.fossil.com/shopping/product/detailmain.jsp?itemID=15116&item%Type=PRODUCT&iMainCat=450&iSubCat=451&iProductID=15116, January 2004. Copy on CD.

[29] Wrist PDA Dress Watch. URL, http://www.fossil.com/shopping/product/detailmain.jsp?itemID=8580&itemT%ype=PRODUCT&iProductID=8580, May 2004. Copy on CD.

[30] A. Birkett. XMMS plugin tutorial at nobugs.org. URL, http://www.xmms.org/docs/vis-plugin.html, April 2004. Copy on CD.

[31] Bluetooth SIG. *Specification of the Bluetooth System*, 1.1 edition, February 2001.

[32] S. Capkun, L. Buttyan, and J.-P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):52 – 65, January - March 2003.

[33] DataCompression Reference Center. RLE - Run Length Encoding. URL, http://web.archive.org/web/20020214085725/http://www.rasip.fer.hr/resea%rch/compress/algorithms/fund/rl/index.html, April 2004. Copy on CD.

[34] J. Fisher and R. Wang. Overview of the Handheld Device Market. URL, http://www.pdamd.com/vertical/features/overmarket.xml, March 2004. Copy on CD.

[35] Free Software Foundation, Inc. *gcc man page*, April 2004.

[36] C. Geiger, B. Kleinnjohann, C. Reimann, and D. Stichling. Mobile AR4ALL. *Augmented Reality Toolkit, The First IEEE [IMG] International Workshop*, page 2 pp., September 2002.

[37] IEEE Standard for Information Technology. *Part11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, September 1999.

[38] Infrated Data Association. *Infrared Data Association Serial Infrared Physical Layer Specification*, May 2001.

[39] N. Kamijoh, T. Inoue, C. M. Olen, M. Raghunath, and C. Narayanswami. Energy trade-offs in the IBM wristwatch computer. In *Proceedings of the 5th International Symposium on Wearable Computing*, pages 133–140, 2001.

[40] R. Ko. Smart Watch: ICW001. URL, http://www.bityard.com/article.php?sid=630, May 2004. Copy on CD.

[41] R. Köpferl. Das Java-Problem. URL, http://dine.mine.nu/webs/koepferl.de/publikationen/java-problem.html, April 2004. Copy on CD.

[42] R. B. Lee. Realtime MPEG video via software decompression on a PA-RISC processor. In *Compcon '95. 'Technologies for the Information Superhighway'. Digest of Papers*, 1995.

[43] T. L. Martin. Time and time again: Parallels in the development of the watch and the wearable computer. In *Proceedings of the 6th International Symposium on Wearable Computers*, 2002.

[44] M. Miller. PDA Review: Fossil Wrist PDA-PC FX2002. URL, http://www.geek.com/hwswrev/pda/fossil/, May 2004. Copy on CD.

[45] Mitsumi. *BluetoothTM Module WML-C09*, April 2004.

[46] C. Narayanaswami, N. Kamijoh, M. Raghunath, T. Inoue, T. Cipolla, J. Sanford, and E. Schlig. IBM's linux watch: The challenge of miniaturization. *IEEE Computer*, pages 33–41, January 2002.

[47] C. Narayanaswami, M. T. Raghunath, and N. Kamijoh. What would you do with a hundred MIPS on your wrist? Technical report, IBM Research Division, Thomas J. Watson Research Center, 2001.

[48] C. Narayanswami and M. Raghunath. Application design for a smart watch with a high resolution display. In *Proceedings of the International Symposium on Wearable Computing*, pages 7–14, Atlanta, Georgia, USA, 2000.

[49] W. Piekarski and B. H. Thomas. An object-oriented software architecture for 3D mixed reality applications. In *In 2nd Int'l Symposium on Mixed and Augmented Reality*, Tokyo, Japan, October 2003.

[50] R. Pollie. Write Once, Run Anywhere–is it for real? URL, http://java.sun.com/features/1997/aug/wora.html, April 2004. Copy on CD.

[51] P. Reisner. DRBD - distributed replicated block device. In *9th International Linux System Technology Conference*, September 2002.

[52] A. J. Richter. *dlopen man page*. Yggdrasil Computing, Inc., 1995.

[53] A. Robertson. Highly-affordable high availability. *Linux Magazine*, November 2003. Copy on CD.

[54] S. Shankland. Yahoo: IBM clocks with new Linux watch. URL, http://linuxtoday.com/it_management/2001032300721PSBZHW, March 2004. Copy on CD.

[55] A. Stiller. Endlich: Waschmaschine am Netz. URL, http://www.heise.de/newsticker/meldung/7171, March 2004. Copy on CD.

[56] J. Strietelmeier. Gadgeteer Hands On Review: Smart Watch ICW001. URL, http://www.the-gadgeteer.com/color-smartwatch-review.html, May 2004. Copy on CD.

[57] J. Strietelmeier and J. Hughes. Gadgeteer Hands On Review: SPOT (Smart Personal Object Technology) Watches. URL, http://www.the-gadgeteer.com/spot-watches-review.html, May 2004. Copy on CD.

[58] I. Sun Microsystems. *PersonalJavaTM Application Environment Specification, Version 1.2a*, November 2000.

[59] Texas Instruments, Post Office Box 655303 Dallas, Texas 75265. *MSP430x13x, MSP430x14x, MSP430x14x1 MIXED SIGNAL MICROCONTROLLER*, April 2004.

[60] E.-W. Toh. Prelude to onHand. URL, http://www.geocities.com/SiliconValley/Sector/3209/onHand1.html, May 2004. Copy on CD.

[61] A. Toney, B. Mulley, B. H. Thomas, and W. Piekarski. Minimal social weight user interactions for wearable computers in business suits. In *Proceedings of the 6th International Symposium on Wearable Computers (ISWC 02)*, 2002.

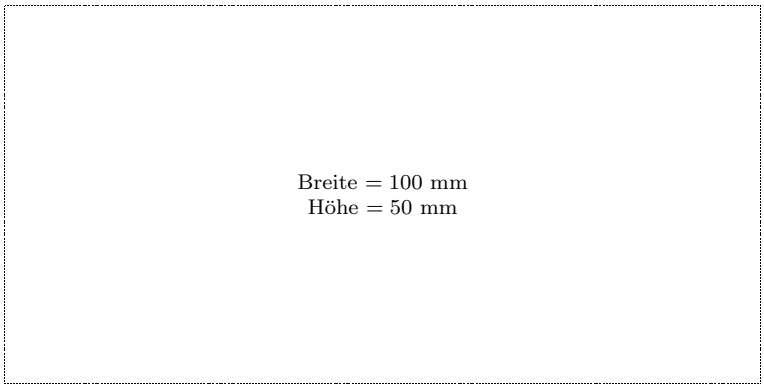
[62] T. Uemukai, T. Hara, M. Tsukamoto, and S. Nishio. A remote display environment: An integration of mobile and ubiquitous computing environments. In *Proceedings of IEEE Wireless Communications and Networking Conference*, pages 618–624, Orlando, Florida, USA, March 2002.

[63] J. T. Vainio. Bluetooth security. URL, http://www.niksula.cs.hut.fi/~jiitv/bluesec.html, June 2004. Copy on CD.

[64] B. Xu, S. Hischke, and B. Walke. The role of ad hoc networking in future wireless communications. In *Proceedings of the ICCT 2003*, April 2003.

Messbox zur Druckkontrolle

— Druckgre kontrollieren! —



— Diese Seite nach dem Druck entfernen! —